SAFETY VERIFICATION
OF MODEL-BASED REINFORCEMENT LEARNING CONTROLLERS
USING REACHABILITY ANALYSIS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Akshita Gupta

In Partial Fulfillment of the

Requirements for the Degree

of

Master of Science

August 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Inseok Hwang, Chair

School of Aeronautics and Astronautics

Dr. Martin Corless

School of Aeronautics and Astronautics

Dr. Arthur Frazho

School of Aeronautics and Astronautics

Dr. Dengfeng Sun

School of Aeronautics and Astronautics

Approved by:

Dr. Wayne Chen

Head of the School Graduate Program

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

SYMBOLS

$\mathcal{X}$	state space subset
$\mathcal{U}$	control space subset
$\mathcal{A}$	control space subset of Player 1
$\mathcal{B}$	control space subset of Player 2
$\mathcal{N}$	bounded noise in the state space
$\mathcal{X}_0$	set of initial states
$\mathcal{V}$	backward reachable set
$\mathcal{W}$	forward reachable set
Υ	initial position of surface interface
Γ	nonanticipative information strategy
$\mathbf{x_t}$	state vector at time t
$\mathbf{x_0}$	initial state vector
$\mathbf{u_t}$	control vector at time t
$\mathbf{a}$	control vector for Player 1
$\mathbf{b}$	control vector for Player 2
$\mathbf{o_t}$	target output in training data set of a neural network
$\hat{\mathbf{o}}$	output of a neural network
π	controller/policy
ψ	unsafe target set
ϕ	viscosity solution of HJI equation
g	functional representation of the initial surface
$W^{(1)}$	weight matrix between input and hidden layer of neural network

$W^{(2)}$	weight matrix between hidden and output layer of neural network
f	system dynamics
f_{NN}	system dynamics approximated by neural network
m	dimension of state vector and noise vector
n	dimension of control vector
t	time

ABBREVIATIONS

HJI Hamilton-Jacobi-Isaac

ML Machine Learning

NN Neural Network

RL Reinforcement Learning

ABSTRACT

Reinforcement Learning (RL) is a data-driven technique which is finding increasing application in the development of controllers for sequential decision making problems. Their wide adoption can be attributed to the fact that the development of these controllers is independent of the knowledge of the system and thus can be used even when the environment dynamics are unknown. Model-Based RL controllers explicitly model the system dynamics from the observed (training) data using a function approximator, followed by using a path planning algorithm to obtain the optimal control sequence. While these controllers have been proven to be successful in simulations, lack of strong safety guarantees in the presence of noise makes them ill-posed for deployment on hardware, specially in safety critical systems. The proposed work aims at bridging this gap by providing a verification framework to evaluate the safety guarantees for a Model-Based RL controller. Our method builds upon reachability analysis to determine if there is any action which can drive the system into a constrained (unsafe) region. Consequently, our method can provide a binary yes or no answer to whether all the initial set of states are (un)safe to propagate trajectories from in the presence of some bounded noise.

1. INTRODUCTION

1.1 Problem Statement

Given a trained Model-Based Reinforcement Learning controller, verify that the controller satisfies all the safety constraints in the state space in the presence of bounded noise.

In this work, safety refers to satisfying the state space constraints at all times. Let $\mathcal{X} \subset \mathbb{R}^n$ denote the set of states and $\mathcal{U} \subset \mathbb{R}^m$ denote the set of feasible controls (actions) that can be executed in any state of the system. Let $f_{NN} : \mathcal{X} \times \mathcal{U} \times \mathcal{N} \rightarrow \mathcal{X}$ be the function that estimates the state transition dynamics of the system when a control $\mathbf{u} \in \mathcal{U}$ is executed in state $\mathbf{x} \in \mathcal{X}$ under the presence of noise $\mathcal{N} \in \mathbb{R}^n$ at time $t \in \mathbb{R}^+$. If $\pi : \mathcal{X} \rightarrow \mathcal{U}$ is the state feedback controller (policy) which returns the optimal action at a given state, then the action returned by following the control law π is denoted as $\mathbf{u}_\pi \in \mathcal{U}$. Similarly, $\mathbf{x}_f(t)$ is defined as the state reached after propagating a trajectory for t time steps from some initial state while following the system dynamics f_{NN} and policy π.

Given an initial set of states $\mathcal{X}_0 \subset \mathcal{X}$ and an unsafe set $\psi \subset \mathbb{R}^n$, we aim to establish

$$\Big|\{\mathbf{x} : \exists \mathbf{x_0} \in \mathcal{X}_0, \exists \mathbf{u}_\pi \in \mathcal{U}, \exists t \in [0, T], \mathbf{x}_f(t) \in \psi\}\Big| \overset{?}{>} 0.$$

If this is a null set then the given controller π is deemed to be safe under bounded noise $\mathcal{N}$ for initial states $\mathcal{X}_0$.

1.2 Motivation

Reinforcement Learning (RL) is a branch of Machine Learning (ML) that finds application in sequential decision making tasks. It is a data driven technique which attempts to "learn" sequential

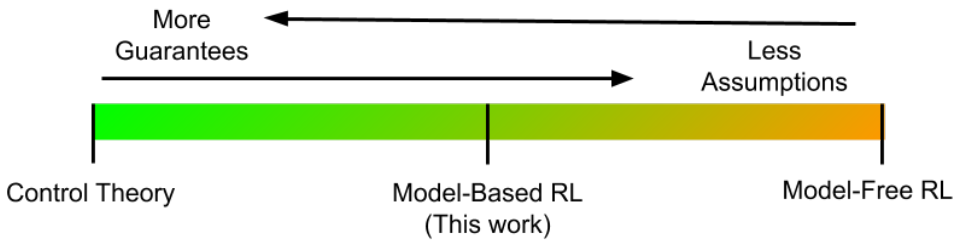

Fig. 1.1. A schematic diagram comparing reinforcement learning to control theory. While control theory aims at providing strong performance guarantees in specific scenarios, RL methods are aimed at general settings but lack strong performance guarantees. In this work, we use Model-Based RL to bridge this gap.

control decisions so as to minimize (maximize) some cost (reward) by directly interacting with the environment and observing the outcomes. It has been increasingly used in the development of controllers for complex tasks, where the complete dynamics of the system is unavailable [1, 2]. The primary reason for its increasing popularity is the fact that such controllers do not depend on the knowledge of the system dynamics, instead they approximate the system dynamics from the data over which they are trained, either explicitly or implicitly. This generality also makes them applicable to systems which require sequential decision making but do not follow a physics based model (e.g. online recommendation systems [3, 4], medical treatment recommendations [5], poker games [6] etc.). Further, for cases where we have partial (full) knowledge of the dynamics, RL methods also provide a potential framework for bridging the best of both the paradigms.

Broadly, RL controllers can be classified as either Model-Based or Model-Free. The main difference between the two types is that the Model-Based controllers try to explicitly model the system dynamics from the training/observed data points and then substitute this model in any planning algorithm to develop the final controller. On the other hand, Model-Free controllers try to learn the control input directly from the training data without modelling the system dynamics. While Model-Free controllers are unaffected by the error in modelling system dynamics, they require a lot more training data and training time. In settings like games and online marketing, where the

data can be obtained in abundance, model-free RL methods have shown significant promise [3, 7]. In comparison, settings where both training data and time can be costly, model-based controllers are more efficient and therefore usually preferred to be deployed on actual systems [8].

Drawbacks

The lack of a definite or approximated model for the system dynamics makes the Model-Free controllers free of any modelling error, but this also makes them a black box technique for which it is hard to give any safety guarantees. Instead, Model-Based controllers are preferred to be deployed on real systems since the approximated model for the system dynamics can be treated as a non-linear system and some formal verification techniques prevalent in control applications can be extended to these controllers.

One of the main issues preventing the deployment of RL controllers on hardware is the difference in the simulation model and the real world model of the system. Most RL controllers are trained to learn a particular behavior on a simulator first which doesn't accurately represent the dynamics of the real world system on which the controller has to be deployed. This discrepancy leads to unexpected behavior after deployment which is not a desirable property for safety critical systems. This was observed in the work Benbrahim and Franklin [9] when they deployed an RL controller on a biped robot to teach it dynamic walking. Similar observations were made in the work of Endo et. al [10] while training a humanoid robot to walk and Morimoto et. al [11] while training a robot to stand up. Most work pertaining to deployment of RL controllers on hardware recognize this problem and usually train the controllers to learn the specific task again on hardware.

Even for Model-Based RL controllers, where the system dynamics is modelled by training a function approximator over a data set of sampled points $(\mathbf{x_t}, \mathbf{u_t}, \mathbf{x_{t+1}})$, the training data may itself be noisy because of the way it is sampled. Ideally, we want to learn only the system dynamics but the sampled data may also contain random noise present in the environment which can be very hard to isolate or remove while extracting the data. This implies that our final controller is not working

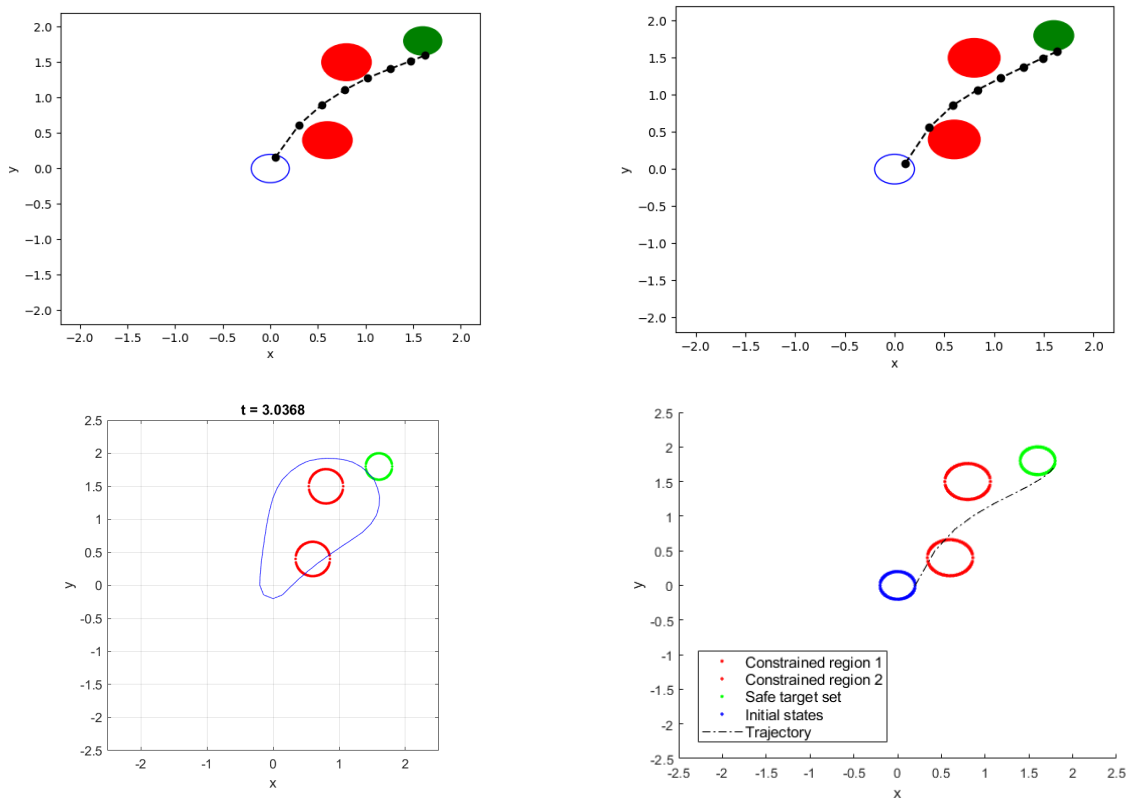

Fig. 1.2. In this thesis, a Model-Based RL Controller is developed to navigate an agent from an initial set of states to a final set of states while avoiding unsafe (constrained) regions. However, RL controllers can not handle hard constraints and an augmented cost function is constructed to train the controller to avoid unsafe regions. The first two figures show randomly sampled trajectories generated while training the RL controller. However, in the third figure, the reachable set is computed for the trained controller and it is observed that the controller fails to meet the safety constraints for all initial states. An example of one such trajectory is presented in the fourth figure.

on the perfect dynamics and once deployed on hardware, the system may very well violate some constraints in the state space. Hence, it is important to predict the controller's performance before it is deployed on hardware.

In this work, we narrow down our focus on Model-Based RL controllers and use techniques prevalent in control applications to provide safety guarantees for such controllers. Specifically, we develop a method to provide safety guarantees for a pre-trained Model-Based RL controller in the presence of bounded noise. This bounded noise represents the modelling error in the system dynamics used to train the controller in simulator and that in the real system. Mathematically, it is incorporated as an additive noise to the state vector. Currently, the proposed method can provide a yes or no answer to whether the given set of initial states will violate any state constraints in the presence of the bounded noise.

Related Work

Providing safety guarantees for RL controllers is a major problem that is being actively looked into by the community. As mentioned above, most RL controllers need to be trained again on hardware to fit the controller to the real system dynamics. Hence, most of the work related to providing safety guarantees is implemented to satisfy state space constraints while training the controller. This has led to the development of switching based safety controllers which switch to a safe controller when the system approaches the boundary of a constraint [12]. Gillula and Tomlin exhibitted safe online learning [13] by developing a switching controller which avoids violating constraints while training using reachablity analysis. Alshiekh et. al monitor the controllers actions before it executes them to make sure that state constraints are not violated [14]. Safe switching controllers have also been developed by following rigorous lyapunov domain knowledge to guarantee minimum performance [15]. This has lead to the concept of barrier certificates which give invariant safe regions with high probability guarantees to prevent a learning agent from violating state constraints [16].

Another approach to analyze the stability of the controller is to empirically verify the lyapunov stability by introducing small bounded noise in the system and observing the systems performance. This method is used in [10] where the authors trained a biped humanoid to learn how to walk and

tested the controller's stability by varying the length of the walking step size and observing that the controller managed to return back to its original gait length when the variation was small. A similar method was adopted in [9] to provide bounds on the inputs for which the biped is able to complete the task of walking.

Other approaches include providing confidence bounds on the expected reward based on sample based planning [17, 18].

The idea of computing reachable sets to provide the safety guarantees of a controller has been used extensively in most control applications and is being extended to RL based controllers as well. Zuo et. al [19] in their work computed the reachable sets for uncertain neural networks with time delays using the Lyapunov-Razumikhin methodology to bound its output space.

1.3 Scope and Goals

This work focuses on developing a method to provide safety guarantees for Model-Based RL Controllers in the presence of bounded noise using reachability analysis. We interpret noise as the modelling error while approximating the system dynamics in the Model-Based controller The following assumptions are made in this work:

- The system operates in the continuous state and action space,

$$\mathcal{X} \subseteq \mathbb{R}^n,\ \mathcal{U} \subseteq \mathbb{R}^m.$$

- A deterministic model of the system dynamics is used in the controller, i.e., given the current state $\mathbf{x_t}$, input $\mathbf{u_t}$ and noise $\mathbf{n_t}$, the dynamics function f_{NN} returns the next state $\mathbf{x_{t+1}}$,

$$f_{NN} : \mathcal{X} \times \mathcal{U} \times \mathcal{N} \rightarrow \mathcal{X}.$$

- The noise $\mathcal{N} \in \mathbb{R}^n$ is assumed to be bounded for all time instances throughout the trajectory,

$$\mathcal{N}_i \in [-d, +d] \ \ i = 1, 2, ..n.$$

- The bounded set of states representing the initial conditions $\mathcal{X}_0 \subset \mathcal{X}$ and final conditions $\mathcal{X}_f \subset \mathcal{X}$ are provided.

Reachable Set Analysis [20] is used to model the evolution of system states for a finite time under noise. This gives a conservative solution determining if there is at least a single state from which the controller can drive the system to an unsafe (constrained) region ψ,

$$\Big|\{\mathbf{x} : \exists \mathbf{x_0} \in \mathcal{X}_0, \exists \mathbf{u}_\pi \in \mathcal{U}, \exists t \in [0, T], \mathbf{x}_f(t) \in \psi\}\Big| \overset{?}{>} 0.$$

Thus, the safety guarantee is not probabilistic in nature, but rather it will give a set of initial states from which the system will never enter an unsafe region under the given bounded noise.

Contributions

Following are the main contributions of this work:

1. The proposed method is independent of the learning/training process used while developing the model-based controller. It is a stand-alone method to evaluate whether any state constraints will be violated when the trained controller is deployed on the hardware in the presence of a bounded noise.

2. This work formulates the learned neural network dynamics in terms of the HJI equation to compute the reachable set.

3. Computing the reachable set using level set method handles the irregular geometric shapes of the reachable set at any time instant, thereby reducing the approximation error.

The choice of using the HJI formulation solved using level set method over other techniques is justified because of its capability of handling both linear and non-linear dynamics. It also paves the way to potentially give performance metrics for the controller in presence of noise, as mentioned in a later section.

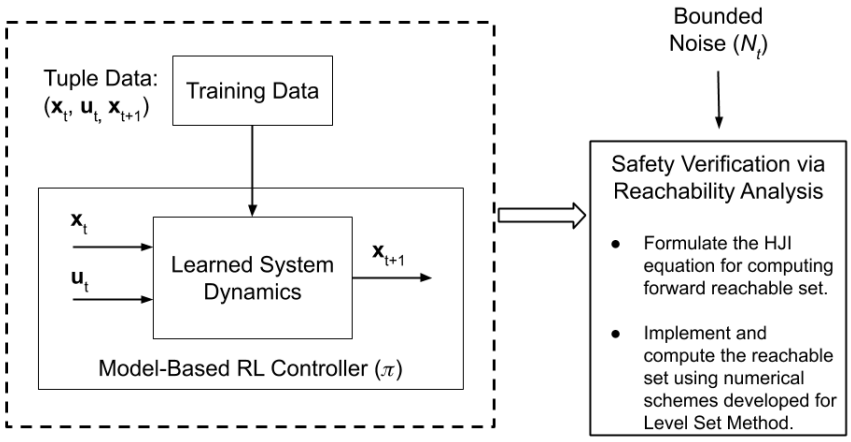

Fig. 1.3. An overview of the methodology.

1.4 Overview

A high level overview of the methodology is presented in Figure 1.3. The proposed method for safety verification is independent of how the controller is developed. It only requires the deterministic model used to approximate the system dynamics f_{NN} and the final control law π. The details for the sequence of steps are given below:

- **Development of the controller**

  1. *Approximating the system dynamics:* The first step towards building a Model-Based RL controller is to approximate the system dynamics by training a universal function approximator (neural network) over a data set. This data set contains many tuples of data, each of which has the state and control (action) taken at time t as input $(\mathbf{x_t}, \mathbf{u_t})$ and the next state $(\mathbf{x_{t+1}})$ as output.

  2. *Developing the controller:* The controller itself is a neural network and its parameters (weights) are trained to maximize the pre-defined reward function (equivalent to the negative of a cost function) over the length of a trajectory.

- **Reachability analysis**

    1. *Formulating the HJI equation:* After developing the controller the problem of forward reachable set is formulated as a HJI equation. To do this, the problem is modelled using game theory where Player 1 is the optimal controller π developed in the previous step and Player 2 is the noise in the system.
    2. *Computing the reachable set:* After the problem formulation, the reachable set is computed using the well developed numerical schemes for level set method. In this work, the toolbox developed by Dr. Ian Mitchell in MATLAB is used [21].

1.5 Thesis Layout

This thesis is structured in the same sequence as described above.

- Chapter 2 describes an overall method to develop a Model-Based RL controller

    1. Section 2.2 gives a detailed explanation on the architecture and training process of a neural network since it is used for approximating the system dynamics and developing the controller.
    2. Section 2.3 introduces Model-Based RL and details the math behind training the controller.
    3. Section 2.4 presents a general algorithm for developing the Model-Based RL controller.
    4. Section 2.5 describes a sample navigation problem. A Model-Based controller is developed for this problem and its results are presented in this section as well.

- Chapter 3 describes the methodology behind reachability analysis

    1. Section 3.1 gives an introduction to reachable sets. It describes the most common algorithms used to compute them and then discusses their applications in safety critical systems.

2. Section 3.2 describes the HJI formulation of reachable sets which is used in this work. Sections 3.2.1-3.2.3 cover in detail the mathematical formulation of the problem while also proving that the design of our Model-Based controller satisfies all the assumptions required for this method.

2. CONTROLLER DESIGN USING MODEL-BASED REINFORCEMENT LEARNING

2.1 Introduction

This chapter looks into designing Model-Based RL controllers for problems requiring sequential decision making. Since the complete system dynamics are not known *a priori* in the problem setting, a universal function approximator - neural network [22], is used to estimate the system dynamics from observed (training) data. The necessary background for building a neural network architecture and estimating its parameters is covered in Section 2.2. Given a trained model of the system dynamics, a parameterized controller is learned using Model-Based RL technique. The relevant background on RL and the method used to develop the controller are presented in Section 2.3 and 2.4, respectively. Once a framework is established, the efficacy of the final controller is demonstrated with an illustrative navigation example which will be used throughout this work. The formulation of this problem and the obtained results are documented in Section 2.5.

2.2 Function Approximation using Neural Networks

A neural network is a parameterized function that is often used to extract patterns from a given data set. The parameters of neural network are used to compose a sequence of non-linear transformations, such that, given an input $\mathbf{x}$, the transformation can be used to map it to the desired output $\mathbf{o}$. An example of one such neural network is presented in Figure 2.1. The first layer is an input layer, where the number of nodes corresponds to the dimension of the input $\mathbf{x}$. Similarly, the final layer is the output layer and the number of nodes is equal to the dimension of output vector

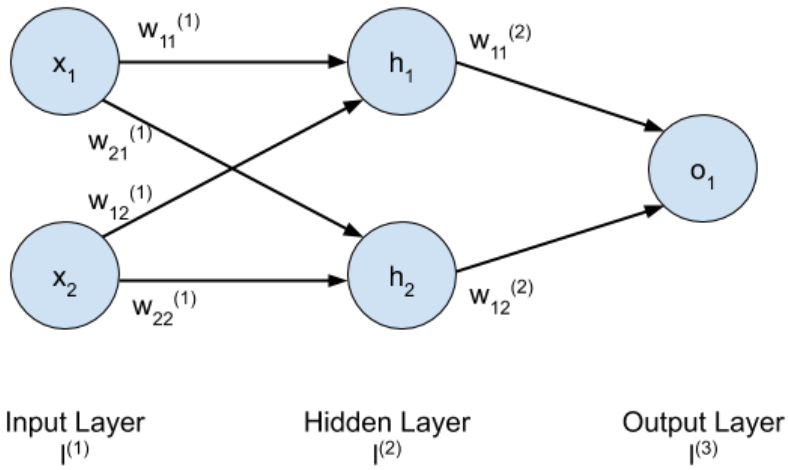

Fig. 2.1. Neural network architecture

o. Additionally, one or more hidden layers can be added between the input and output layers (for example, Figure 2.1 contains one hidden layer with two nodes).

Using a neural network involves three prime considerations: (1) Architecture design: This corresponds to deciding the number of parameters to use and the corresponding functional form of the network, for example, number of hidden nodes, number of hidden layers, etc. (2) Feedforward pass: Once the architecture design has been decided, the procedure of using the network to transform an input **x** to an output **o** is known as the feedforward pass. (3) Backpropagation: given a functional form of the network, we want to estimate its parameter values that would let it best approximate the desired transformation. In the following paragraphs, we elaborate on each of these three points.

1. **Architecture:** We consider a simple neural network model with 2 nodes in the input layer, 2 nodes in the hidden layer and 1 node in the output layer as shown in Figure 2.1. Each node in one layer is connected to each node in the next layer through a scalar weight. For convenience, these weights are represented in the matrix form. Let there be p, q and r number

of nodes in the input, hidden and output layer, respectively. Then the weight matrix between the input and hidden layer is defined as $W^{(1)} \in \mathbb{R}^{q \times p}$ and that between the hidden layer and output layer is defined as $W^{(2)} \in \mathbb{R}^{r \times q}$. These weights are randomly initialized and are updated throughout the training process to better fit the data. The terminology followed in this thesis defines w_{ab}^{l} as the weight between node b in layer l and node a in layer $l+1$. Thus w_{11}^{2} denotes the weight between node 1 in layer 2 and node 1 in layer 3. Thus the weights in Figure 2.1 can be put in terms of matrices as:

$$W^{(1)} = \begin{bmatrix} w_{11}^{(1)} & w_{12}^{(1)} \\ w_{21}^{(1)} & w_{22}^{(1)} \end{bmatrix}$$

$$W^{(2)} = \begin{bmatrix} w_{11}^{(2)} & w_{12}^{(2)} \end{bmatrix}$$

Each cycle of training a neural network constitutes the feedforward pass followed by back-propagation over the entire training data.

2. **Feedforward Pass:** In the feedforward pass, the neural network predicts the output for a given input vector. First, we need to define how a node is computed in the layers following the input layer. For an input vector $\mathbf{x} \in \mathbb{R}^p$, the nodes in the hidden layer $\mathbf{h} \in \mathbb{R}^q$ and output layer $\hat{o} \in \mathbb{R}$ are computed as:

$$\mathbf{h} = f_1(W^{(1)}\mathbf{x}), \tag{2.1}$$

$$\hat{o} = f_2(W^{(2)}\mathbf{h}) = f_2(W^{(2)} f_1(W^{(1)}\mathbf{x})), \tag{2.2}$$

where $f_1()$ and $f_2()$ are called the activation functions which are used to model the non-linear part of the training data. In general, the output of neural network with $n-1$ hidden layers can be written as:

$$\hat{\mathbf{o}} = f_n(W^{(n)} f_{n-1}(W^{(n-1)} f_{n-2}(...))). \tag{2.3}$$

3. **Backpropagation:** In backpropagation, a gradient-based method is used to update the weights of the neural network to increase its prediction accuracy over the training data. The training

data is a tuple which contains both the input $\mathbf{x}$ and the corresponding target output $\mathbf{o}$. In Figure 2.1, the target output is a scalar o. Let

$$E = \frac{1}{n}\sum_{i=1}^{n}(o_i - \hat{o}_i)^2$$

be the mean square error between the target output o_i and the predicted neural network output $\hat{o}_i$ over n training samples. The update rule for the weights is given as

$$W_{t+1}^{(1)} = W_t^{(1)} - \alpha\frac{\partial E}{\partial W_t^{(1)}},$$

$$W_{t+1}^{(2)} = W_t^{(2)} - \alpha\frac{\partial E}{\partial W_t^{(2)}},$$

where α is the learning rate. It can also be time varying in which case it is written as α_t. The time subscript refers to the number of times the weight matrix has been updated. To make the notation simpler, it has been dropped in the following equations.

The partial derivatives in the update rule can be computed by using the chain rule as follows:

$$\begin{aligned}\frac{\partial E}{\partial W^{(2)}} &= \frac{\partial E}{\partial \hat{o}}\cdot\frac{\partial \hat{o}}{\partial W^{(2)}} \\ &= \frac{2}{n}\sum_{i=1}^{n}(o_i - \hat{o}_i)\cdot f_2'(W^{(2)}\mathbf{h})\mathbf{h},\end{aligned} \tag{2.4}$$

$$\begin{aligned}\frac{\partial E}{\partial \mathbf{h}} &= \frac{\partial E}{\partial \hat{o}}\cdot\frac{\partial \hat{o}}{\partial \mathbf{h}} \\ &= \frac{2}{n}\sum_{i=1}^{n}(o_i - \hat{o}_i)\cdot f_2'(W^{(2)}\mathbf{h})W^{(2)},\end{aligned} \tag{2.5}$$

$$\begin{aligned}\frac{\partial E}{\partial W^{(1)}} &= \frac{\partial E}{\partial \hat{o}}\cdot\frac{\partial \hat{o}}{\partial \mathbf{h}}\cdot\frac{\partial \mathbf{h}}{\partial W^{(1)}} \\ &= \frac{2}{n}\sum_{i=1}^{n}(o_i - \hat{o}_i)\cdot f_2'(W^{(2)}\mathbf{h})W^{(2)}\cdot f_1'(W^{(1)}\mathbf{x})\mathbf{x} \\ &= \frac{\partial E}{\partial \mathbf{h}}\cdot f_1'(W^{(1)}\mathbf{x})\mathbf{x}.\end{aligned} \tag{2.6}$$

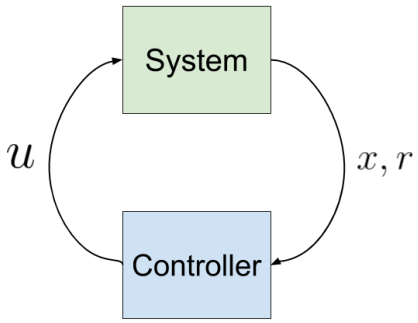

Fig. 2.2. In the reinforcement learning paradigm, the controller observes the current state x and executes an action u. Based on this, the system transitions accordingly. The controller then receives a feedback r for the decision taken and also observes the next state.

2.3 Model-Based Reinforcement Learning

Reinforcement Learning (RL) is a branch of Machine Learning (ML) that aims at finding a controller (policy) to make sequential decisions that minimizes the total cost (maximizes the total reward). RL is a data driven technique that doesn't assume any knowledge of the system dynamics but a way to sample trajectories in the environment. This makes RL algorithms generally applicable to a broad range of problems, ranging from sequential medical treatment [5], online recommender systems [3, 4], game playing [6, 7], robotics [8] etc.

A typical setup for the RL paradigm is illustrated in Figure 2.2. On observing a system state **x**, the controller executes an action **u**. Consequently, the controller receives a feedback r and also observes the next state. The goal of the controller is to learn a sequence of control decisions, such that it can minimize (maximize) the total sum of cost (reward) incurred.

[NOTE: In Reinforcement Learning terminology, the feedback provided to the agent is called the reward r. This is nothing but the negative of the cost function.]

Broadly, RL methods can be grouped as either model-free or model-based. Model-free methods learn the controller without trying to explicitly approximate the system dynamics [23, 24].

Such methods can avoid controllers from getting biased due to the errors in modelling the system dynamics. Their general nature implies that very little is assumed about the underlying structure of the problem and thus very little domain knowledge can be pre-engineered in such methods. Being data-driven, such methods therefore rely upon significant amount of data and hyper-parameter tuning to solve any particular problem.

In contrast, model-based methods aim at explicitly modelling the state transition dynamics using the transition (training) samples. Methods built on these types of controllers make use of universal function approximators like neural networks to learn the system dynamics accurately. Once the system dynamics are learned, the estimated model can then be used to learn the controller in a more sample efficient manner. Hence, from here on, we restrict our focus to Model-Based RL. In the following sections, we first formalize the setting of Model-Based RL. Then we establish the method for estimating the parameters of the controller using the learned dynamics model. Finally, we combine all the above concepts and present the complete algorithm for obtaining both the estimated model and the controller.

2.3.1 System Definition

For the Model-Based RL we consider the setup where f_{NN}, a differentiable model of the system dynamics, is learned using observed data and provided. A controller π_θ, parameterized using the weights θ, observes the state $x_t \in \mathcal{X}$ and executes the control $u_t \in \mathcal{U}$, i.e.,

$$u_t = \pi_\theta(x_t).$$

The next state is then obtained using

$$x_{t+1} = f_{NN}(x_t, u_t) + \mathcal{N}_{t+1},$$

where $\mathcal{N}_{t+1} \subset \mathbb{R}^m$ is the noise term. The feedback $r_{t+1} \in \mathbb{R}$ is obtained as,

$$r_{t+1} = R(x_{t+1}).$$

A visual depiction of a rolled out trajectory using the above procedure is provided in Figure 2.3.

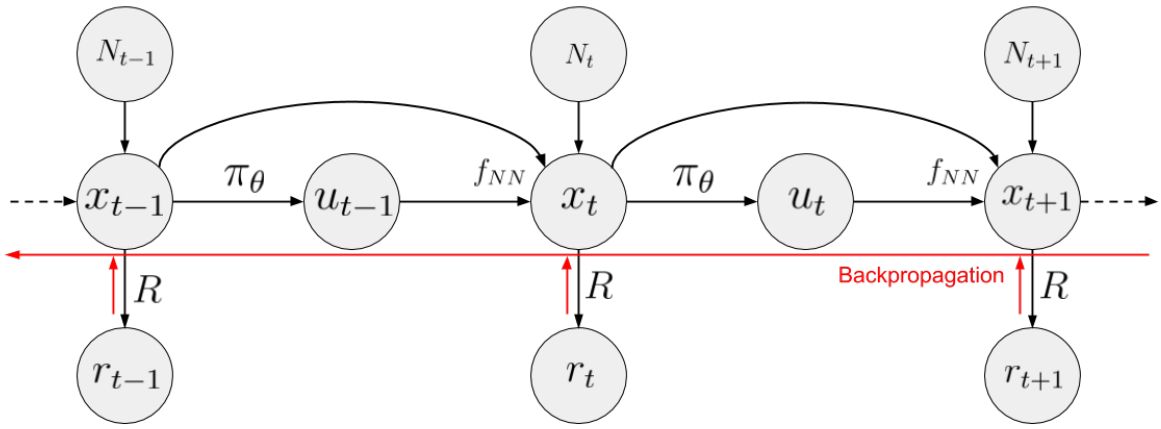

Fig. 2.3. Simulated trajectory using the learned dynamics model. The policy π_θ selects control u_t for a given state x_t. The next state x_{t+1} is then obtained using the transition dynamics model f_{NN}, under the influence of noise $\mathcal{N}_t$. Reward r_t for each state is obtained using the function R for each state x_t. The red arrows indicate the backpropagation of gradients use to maximize the total sum of rewards.

2.3.2 Controller's Parameters Estimation

This section develops the procedure to estimate the parameters of the controller such that it can be used to maximize (minimize) the reward (cost). The derivation given below is specific to the way we are developing our controller. For a more general understanding of value and policy gradient methods the reader can refer to [25].

Let the total reward over the trajectory be represented as $J(\theta)$, such that

$$J(\theta) = \mathrm{E}\left[\sum_{t=0}^{T} r_t \middle| \pi_\theta, f_{NN}\right], \tag{2.7}$$

where conditioning on π_θ and f_{NN} means that the action is chosen using the policy π_θ and the next state is simulated using the learned model f_{NN}. Note that although both π_θ and f_{NN} are deterministic, the expectation arises due to the stochastic noise $\mathcal{N}_t$ affecting the state transition at every time step t in the trajectory. To maximize $J(\theta)$, we seek to iteratively update the parameters θ by ascending the gradient of $J(\theta)$,

$$\theta \leftarrow \theta + \alpha \frac{\partial J(\theta)}{\partial \theta}. \tag{2.8}$$

However, it is not immediately evident how to compute $\frac{\partial J(\theta)}{\partial \theta}$. In what follows, we demonstrate how it can be computed in terms of the differentiable functions π_θ, f_{NN} and R.

As $J(\theta)$ is the sum of all the rewards r_t,

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial r_t}{\partial \theta}.$$

Using the fact that $r_t = R(x_t)$,

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \cdot \frac{\partial x_t}{\partial \theta}.$$

Since observing x_t is a consequence of all the controls u_i, executed using π_θ, in the previous time steps $i \in [0, t]$,

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \sum_{i=0}^{t} \frac{\partial x_t}{\partial u_i} \cdot \frac{\partial u_i}{\partial \theta}.$$

Using the fact that $u_t = \pi_\theta(x_t)$,

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \sum_{i=0}^{t} \frac{\partial x_t}{\partial u_i} \cdot \frac{\partial \pi_\theta(x_i)}{\partial \theta}.$$

Using the chain rule, the derivative of observing state x_t with respect to u_i can be rewritten using the intermediate variable x_{i+1},

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \sum_{i=0}^{t} \frac{\partial x_t}{\partial x_{i+1}} \cdot \frac{\partial x_{i+1}}{\partial u_i} \cdot \frac{\partial \pi_\theta(x_i)}{\partial \theta}.$$

As $x_{t+1} = f_{NN}(x_t, u_t)$,

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \sum_{i=0}^{t} \frac{\partial x_t}{\partial x_{i+1}} \cdot \frac{\partial f_{NN}(x_i, u_i)}{\partial u_i} \cdot \frac{\partial \pi_\theta(x_i)}{\partial \theta}.$$

Again, using the chain rule, the derivative of x_t with respect to x_{i+1} can be written using the intermediate states x_{i+1} to x_t,

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \sum_{i=0}^{t} \left(\prod_{k=i+1}^{t-1} \frac{\partial x_{k+1}}{\partial x_k} \right) \cdot \frac{\partial f_{NN}(x_i, u_i)}{\partial u_i} \cdot \frac{\partial \pi_\theta(x_i)}{\partial \theta}.$$

As the next state is the function of two variables - the current state and the action, we get two corresponding terms for the derivative of x_{k+1} with respect to x_k,

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \sum_{i=0}^{t} \left(\prod_{k=i}^{t-1} \left(\frac{\partial f_{NN}(x_k, u_k)}{\partial x_k} + \frac{\partial f_{NN}(x_k, u_k)}{\partial u_k} \cdot \frac{\partial u_k}{\partial x_k} \right) \right) \cdot \frac{\partial f_{NN}(x_i, u_i)}{\partial u_i} \cdot \frac{\partial \pi_\theta(x_i)}{\partial \theta}.$$

Finally, observe that $u_k = \pi_\theta(x_k)$, therefore

$$\frac{\partial J(\theta)}{\partial \theta} = \sum_{t=0}^{T} \frac{\partial R(x_t)}{\partial x_t} \sum_{i=0}^{t} \left(\prod_{k=i}^{t-1} \left(\frac{\partial f_{NN}(x_k, u_k)}{\partial x_k} + \frac{\partial f_{NN}(x_k, u_k)}{\partial u_k} \cdot \frac{\partial \pi_\theta(x_k)}{\partial x_k} \right) \right) \cdot \frac{\partial f_{NN}(x_i, u_i)}{\partial u_i} \cdot \frac{\partial \pi_\theta(x_i)}{\partial \theta}.$$

2.4 Algorithm

The core algorithm can be expressed in four major steps:

- Collect transition samples (x, u, x') using a random policy.

- Estimate the system dynamics using $f_{NN}(x, u)$ by minimizing $\sum_i \|f_{NN}(x_i, u_i) - x_i'\|^2$.
- Estimate parameters θ for the controller π_θ by maximizing $J(\theta)$.
- Execute the control $u = \pi_\theta(x)$ for any x.

2.5 MATLAB Simulation

2.5.1 Example Problem

The following test problem is used throughout this work to demonstrate the algorithm.

We select a problem where the system dynamics are known so that it is easier to check the accuracy achieved by the trained neural network. However, in the implementation of the controller, no knowledge of the actual dynamics is assumed and only the dynamics modelled by the neural network is used.

The selected problem has two constraints in the state space (2.11) and a set of feasible initial conditions (2.12) and final conditions (2.13). Figure 2.4 illustrates the setting of this test problem. The controller is trying to maximize the reward over controls V and θ at each time step. This reward function is presented in (2.9). The $dist()$ function determines whether a given state falls under a circular region or not. The first term in the reward corresponds to the final target set and the second and third terms correspond to the constraints. These constraints cannot be handled directly by the neural network policy and have to be incorporated in the reward function itself.

$$\begin{aligned} Maximize_{V,\theta} \sum_{t=1}^{t_f} \{ & -1.2 * dist([x_{t+1}, y_{t+1}], [1.6, 1.8], 0.04) \\ & +0.2 * dist([x_{t+1}, y_{t+1}], [0.6, 0.4], 0.26) \\ & +0.2 * dist([x_{t+1}, y_{t+1}], [0.8, 1.5], 0.26) \\ & -0.1\} \end{aligned} \tag{2.9}$$

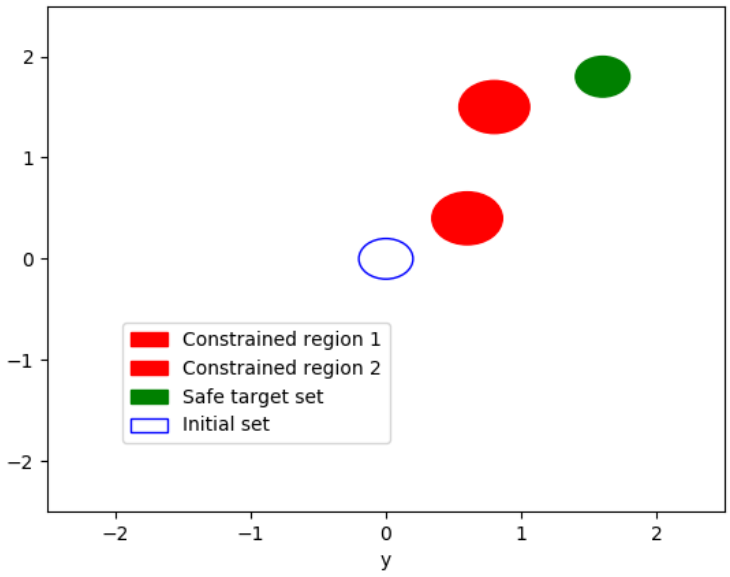

Fig. 2.4. This figure shows the setting of the test problem. There are 2 constrained regions which the controller must avoid while navigating from a randomly selected state from the initial set to the safe target set.

subject to

$$\begin{aligned} \frac{dx}{dt} &= Vcos(\theta), \\ \frac{dy}{dt} &= Vsin(\theta) \end{aligned} \tag{2.10}$$

and constraints

$$\begin{aligned} (x-0.6)^2 + (y-0.4)^2 >= 0.26, \\ (x-0.8)^2 + (y-1.5)^2 >= 0.26 \end{aligned} \tag{2.11}$$

with boundary conditions

$$x_0^2 + y_0^2 \leq 0.04, \tag{2.12}$$

$$(x_f - 1.6)^2 + (y_f - 1.8)^2 \leq 0.04. \tag{2.13}$$

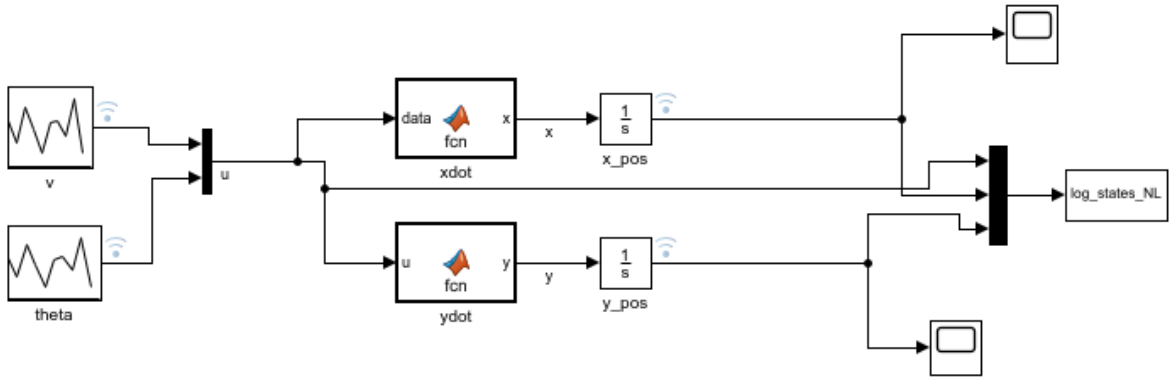

Fig. 2.5. Simulink model used to generate training data

2.5.2 Data Collection

To generate the data tuples required to train the neural network, we developed a simulink model of the system, as shown in Figure 2.5, and propagated trajectories for 10 seconds with random actions sampled at the rate of 0.1 seconds. Note that these random actions may take the vehicle in the constrained regions of our test problem but this can be ignored because we ultimately want to learn the dynamics of the agent irrespective of the structure of the environment. Generating data while ignoring the constraints doesn't compromise the integrity of the proposed algorithm and makes the generation of training data easier. Practically speaking, the training data is usually provided to the designer of the controller.

In the simulink model, there are two random source blocks for the control variables V and θ. The properties of these source blocks are listed in Table 2.1. The bounds on the velocity V are selected arbitrarily. These inputs are combined into a single signal line using the Mux block and are then fed into the function blocks xdot and ydot which define the system dynamics. The definitions of the two function blocks corresponding to the x and y positions are presented in Table 2.2. Each of these function blocks are followed by an integrator block where the initial values of x

Table 2.1.
Input variables as described in the simulink model

Name	Source Type	Min Value	Max Value	Sample Time
theta	Uniform	0	2*pi	0.1
V	Uniform	0	10	0.1

Table 2.2.
Function blocks which describe the system dynamics

Name	Script

xdot	`function x = fcn(data)`
	`  x = data(1)*cos(data(2));`
	`end`

ydot	`function y = fcn(u)`
	`  y = u(1)*sin(u(2));`
	`end`

Table 2.3.
Details about the number of sampling points used to generate training data. Each trajectory is propagated from a random initial point $[x_0 \;\; y_0] \in [-0.2 \;\; 0.2]$

Sample Points per Trajectory	Number of Trajectories	Total Number of Data Points
100	10	1000

and y are defined. The output from these integrator blocks gives the next state of the system. This output is logged in a log file.

A total of 10 trajectories are propagated with the initial values for x and y selected randomly from $[-0.2 \;\; 0.2]$. Thus, there are 1000 tuples of data generated.

2.5.3 Approximated System Dynamics

Out of 1000 tuples of data, 80% are randomly chosen to train the neural network and the remaining 20% are used as testing data. It is important to monitor the error for both training and testing to check if the neural network is over-fitting the training data. Ideally, the testing and training error should be close. But if the training error is much lower than the testing error, it can be concluded that the neural network is over-fitting the training data and will perform poorly when new data points are given as inputs.

According to the problem formulation in section 2.5.1, there are 4 nodes in the input layer $\mathbf{x_i} = [x_t, y_t, V_t, \theta_t]^T$ and 2 nodes in the output layer $\mathbf{o_i} = [x_{t+1}, y_{t+1}]^T$. The activation function used for computing both the hidden layer nodes and output layer nodes is $tanh$. The output is then scaled to fit the dimensions of the arena in the test problem.

There are 14 nodes in the hidden layer. This number was fixed after training the neural network with different number of nodes in the hidden layer and observing the error in approximation. The plots for the error profiles are shown in Figures 2.6 and 2.7. Neural networks with hidden nodes 12, 14 and 16 have the lowest training error and that with 14 nodes has the lowest testing error. Table 2.4 gives the time taken to train these neural networks. For the test problem there is a difference of only a few seconds between training the different architectures. However, for more complex problems the training time can also be an important factor in determining the final architecture of the neural network.

Table 2.4.
Time taken (in seconds) to train neural networks with different number of nodes in the hidden layer.

Number of nodes	2	4	6	8	10	12	14	16
Training time (sec)	152.65	155.25	160.52	162.41	136.98	166.03	185.93	189.65

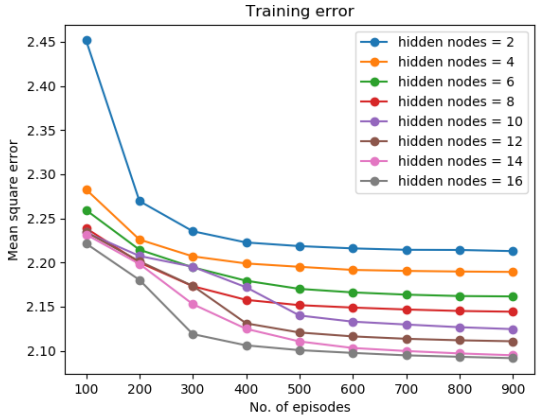

Fig. 2.6. Training error over 900 episodes for neural networks with different number of nodes in the hidden layer.

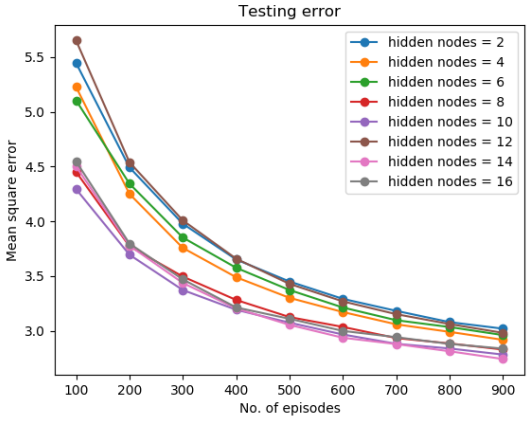

Fig. 2.7. Testing error for neural networks with different number of nodes in the hidden layer.

2.5.4 Results

The final Model-Based controller is a state feedback controller. It is also modelled as a neural network which has 4 nodes in the input layer corresponding to the current state $[x_t, y_t]$ and inputs at previous time step $[V_{t-1}, \theta_{t-1}]$, 14 nodes in the hidden layer and 2 nodes in the output layer corresponding to optimal controls $[V_t, \theta_t]$ that should be executed in the given state. More details regarding the controller architecture is given in table 2.5. The results were obtained after training the controller for 5,000 episodes in the presence of a random Gaussian noise with mean 0 and variance $1e-2$. The hyper parameters for the controller were set after multiple trials (table 2.6) and the results are reproducible. The performance of the controller with random initial points can be seen in Figure 2.5.4 below.

Table 2.5.
Neural network architecture for the model-based controller.

Nodes in input layer	**Activation function between input layer and hidden layer**	**Nodes in hidden layer**	**Activation function between input layer and hidden layer**	**Nodes in output layer**	**Bounds on the controls**
4	$tanh()$	14	$V : sigmoid()$ $\theta : tanh()$	2	$[0\ 0.1]$ $[-\pi\ \pi]$

Table 2.6.
Hyper-parameter values for the model-based controller.

No. of training episodes	**Learning rate**	**Noise**	**Seed**
5,000	$2e-3$	$N(0, 1e-2)$	12345

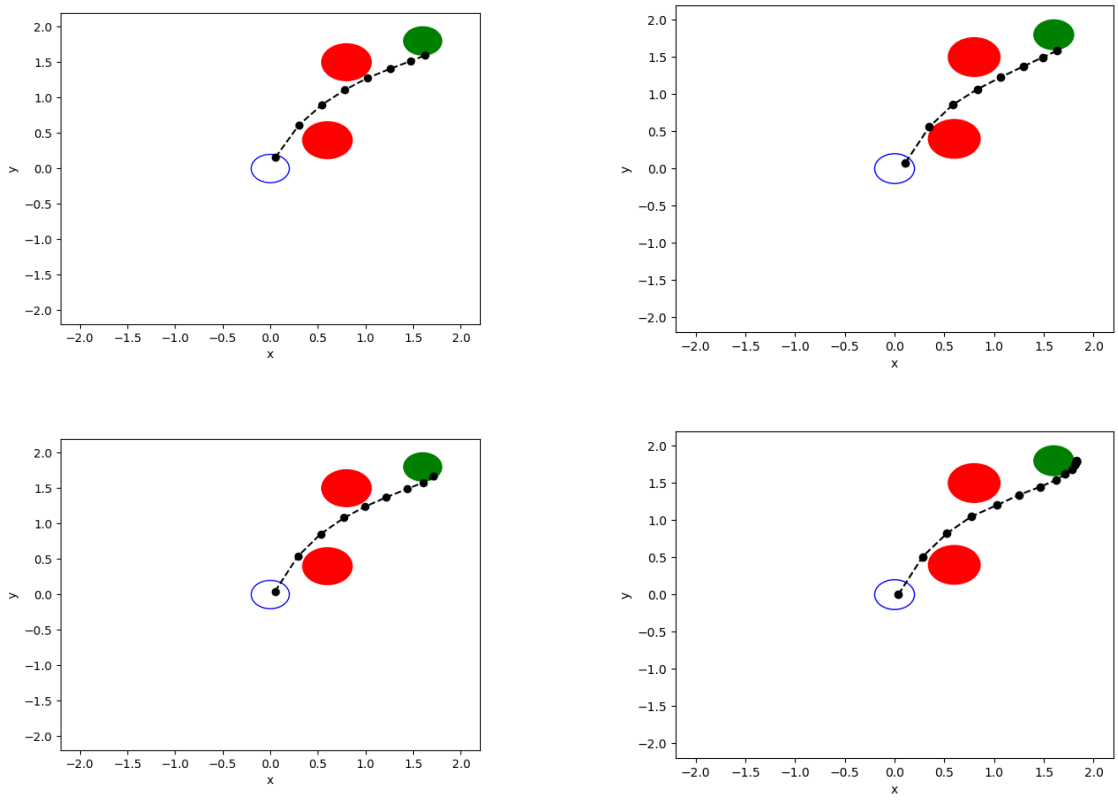

Fig. 2.8. Sample trajectories generated by the trained model-based RL controller from random initial conditions

3. METHODOLOGY

This chapter describes the methodology used to validate the safety guarantees for a Model-Based RL controller. We use reachability analysis on a trained Model-Based controller in the presence of bounded noise to monitor the evolution of system states for a finite time. If at any time, this reachable set intersects with an unsafe (constrained) region, then the controller is not safe to operate in the presence of the bounded noise for the given set of initial states.

Section 3.1 introduces the concept of reachable sets. Section 3.2 justifies and describes the choice of reachable set algorithm used in this work - the Hamilton-Jacobi-Isaac Formulation using the Level Set Method.

3.1 Reachable Sets

The reachable set is a formal verification method which has been used extensively in discrete, continuous and hybrid systems to provide safety guarantees. The idea behind reachable set has been derived from the technique of exhaustive testing/validation. It attempts to propagate a set of trajectories at each time step to encompass the evolution of all the system states with time. The control input is selected with respect to the cost function which it is trying to minimize or maximize.

However, it is impossible to compute the trajectories for all possible states in an uncountable continuous domain. Even in a discrete domain, the curse of dimensionality prevents the computation of an exhaustive set of trajectories. To reduce this computational cost, most methods try to approximate the reachable set instead and give a conservative solution. Before delving into different approximation schemes, we give a mathematical definition of reachable sets.

3.1.1 Definition

For a continuous system with n states and m actions, which is defined to operate in a state space $\mathcal{X} \in \mathbb{R}^n$, we define $\mathcal{X}_0 \subset \mathcal{X}$ as its initial set of states and an input set $\mathcal{U} \in \mathbb{R}^m$ which is defined over the entire state space. The state vector at any time $t \in \mathbb{R}^+$ is denoted as $\mathbf{x} \in \mathcal{X}$ and the input vector as $\mathbf{u} \in \mathcal{U}$. This system follows the system dynamics

$$\dot{x} = f(x, u). \tag{3.1}$$

If the input follows the control law stated by the controller $\pi : \mathcal{X} \to \mathcal{U}$, it is denoted as $\mathbf{u}_\pi \in \mathcal{U}$. Similarly, $\mathbf{x}_f(t)$ is defined as the state reached after propagating a trajectory for t time steps while following the system dynamics f and controller π.

Given an unsafe set $\psi \subset \mathbb{R}^n$, the reachable set determines

$$\Big|\{\mathbf{x} : \exists \mathbf{x_0} \in \mathcal{X}_0, \exists \mathbf{u}_\pi \in \mathcal{U}, \exists t \in [0, T], \mathbf{x}_f(t) \in \psi\}\Big| \overset{?}{>} 0.$$

3.1.2 General Approach

The idea behind computing reachable sets can be expressed abstractly. As stated in [26], the semigroup property of dynamical systems enables us to compute the reachable sets iteratively.

$$Reach_{[0,t_2]}(X_0) = Reach_{[0,t_2-t_1]}(Reach_{[0,t_1]}(X_0)) \tag{3.2}$$

for $t_2 > t_1 > 0$

In (3.2), we define a function $Reach_{[0,t]}(X)$ which computes the reachable set by propagating the states in $\mathcal{X}$ for a time interval t following the system dynamics in (3.1). Using this function the abstract algorithm below details a general method for computing reachable sets which can be tailored to fit a particular problem [27, 28]. If we want to compute N intermediate reachable sets over a time interval $[0\ T]$, we integrate the system dynamics every r time steps where $r = \frac{T}{N+1}$.

Algorithm 1: Algorithm for computing reachable set

Input: $X_0 \subset X$

Output: $Q = Reach_{[0,T]}(X_0)$

```
P := Q := X_0
r = T/(N + 1)
for i = 0,....N do
    P := Reach_[0,r](P)
    if P ⊆ Q then
        break
    else
        Q := Q ∪ P
    end
end
```

In the above algorithm, we start propagating the system states from the initial conditions X_0 for a finite time T. If at any time $t \in [0\,T]$ the reachable set intersects with an unsafe region in the state space, we conclude that for the given initial conditions there exists at least one control input which can drive the system to an unsafe/constrained region. Since we start from the initial conditions, we can only comment on whether or not the system will violate the constraints at any time throughout its trajectory. However, a more useful result will be to explicitly compute the set of initial states which can lead the system to the unsafe region. For this, we start propagating the trajectories backwards from the unsafe target region for time $[-T\ 0]$ and the intersection of this reachable set with the initial set will give the unsafe initial states which should be avoided. The former method is said to compute the forward reachable set, while the latter computes the backward reachable set.

It is evident from the above algorithm that during implementation, the representation of a reachable set should be conducive to set operations. This is not a trivial task. For discrete systems, the problem lies in the large storage space required to keep track of these sets [27]. For continuous systems, appropriate geometric interpretations of the sets have to be formulated which can define

the boundaries of the reachable set and over which it is easy to perform union and intersection set operations. This is challenging because reachable sets often don't follow regular well-defined geometric shapes.

To counter these problems, most algorithms approximate the reachable sets instead of computing them exactly. This gives a conservative solution to the problem. Care should be taken as to when we should over-approximate or under-approximate the reachable set. For example, when a backward reachable set is computed for an unsafe target region it should be over-approximated to provide a conservative solution (Figure 3.1). Similarly, if a backward reachable set is computed for a safe target set then it should be under-approximated to provide a conservative solution (Figure 3.2).

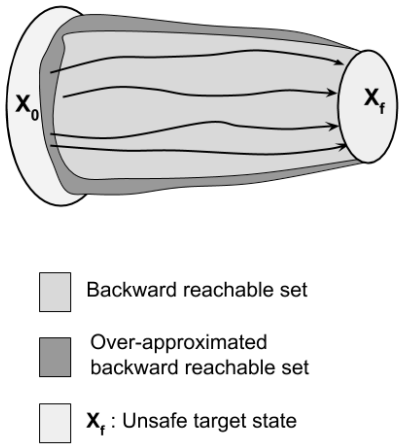

Fig. 3.1. Over-approximation of the backward reachable set to find the initial set of states leading to an unsafe target set.

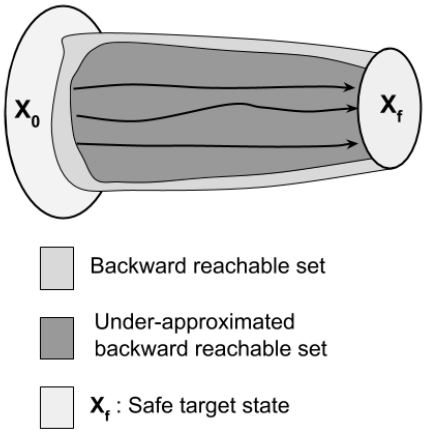

Fig. 3.2. Under-approximation of the backward reachable set to find the initial set of states leading to a safe target set.

Geometric structures like polyhedrons [29, 30, 31], ellipsoids [32, 33, 34] and hyperplanes [35, 36] have been used to approximate the reachable set. To overcome the problem of computational cost, several works have explored the idea to decompose the original system into a lower

dimensional space [37, 38] or subsystems [39] and then compute the approximate reachable set. While these methods make the computation more efficient, if the approximation error is large, it compounds over time and the resulting solution may be unusable.

3.1.3 Applications

Reachable sets are primarily used to provide safety guarantees for discrete, continuous and hybrid controllers. Apart from this, they have been successfully used for developing path planning algorithms [40, 41] and collision avoidance algorithms [42, 43, 44]. Their application extends over multi-vehicle systems as well and they have been successfully used to provide safety guarantees for multiple aerial vehicle systems [45, 46] and incorporate collision avoidance in such systems [47].

Reachable set is not a new concept for machine learning controllers either. Recently, safe online learning controllers [13, 48] have been developed using reachability analysis.

In this thesis we use the Hamilton-Jacobi-Isaac (HJI) - formulation based level set method to compute the reachable sets to estimate the performance of a model-based RL controller in presence of noise.

3.2 Reachable Set - Hamilton-Jacobi-Isaac Formulation

This work uses the Hamilton-Jacobi-Isaac (HJI) Formulation of the Level Set Method to compute the reachable set [20]. The choice of this particular method is justified below:

1. This method is applicable to both linear and non-linear systems.

2. It can work on any geometric form adopted by the reachable set, i.e., it doesn't approximate an irregularly shaped reachable set into a well defined one. This reduces the approximation error.

3. The method uses the already well-defined numerical tools for the level set methods, making the implementation easier, sound and efficient.

To reiterate the problem, we want to use reachable sets to provide safety guarantees for a trained Model-Based RL controller in the presence of bounded noise $\mathcal{N}(t) \in \mathbb{R}^n$. In this work, noise represents the modelling error while training a neural network to represent the system dynamics. It is additive noise to the system.

$$\mathcal{N}_i \in [-d \quad +d], \quad i = 1, 2...n.$$

This problem is practically important because the modelling error implies that the controller is not operating on the correct real-world system dynamics and a preliminary analysis of the controller's behavior when operating in the presence of this noise will prevent any unexpected behavior when it is deployed on hardware.

3.2.1 Game Theory Problem Formulation

This work uses the HJI formulation for computing the reachable set as presented in [20]. Thus, for detailed proofs of the theorems used in the following sections the reader can refer to [20]. It should be noted here that the original paper computed the backward reachable set and hence, all the theorems were proved using the backward reachable set formulation. However, since then, many works have noted that these theorems are valid for forward reachable sets as well [49, 50, 51]. We need to be careful while defining the set of states for backward and forward reachable set.

Thus, following the notation in [49], a backward reachable set computed for a finite time t is denoted as

$$\mathcal{V}(t, t_f), \tag{3.3}$$

where, $t < t_f$.

A forward reachable set computed for a finite time t is denoted as

$$\mathcal{W}(t_0, t), \tag{3.4}$$

Table 3.1.
Objectives of Player 1 and Player 2 for safe and unsafe target sets

	Unsafe Target Set	Safe Target Set
Player 1 (Optimal Controller)	Maximize cost	Minimize cost
Player 2 (Noise/Adversary)	Minimize cost	Maximize cost

where, $t > t_0$.

In this work, we compute the forward reachable set and following the HJI formulation in [20], we formulate the problem in terms of game theory. We begin by modelling the problem of computing the forward reachable set as a differential game between 2 players. Player 1 is the model-based RL controller which wants to drive the agent/plant towards its final state (safe target set). Player 2 is the adversary and represents the noise which wants to drive the agent/plant away from its final state. If a cost function is defined as the distance between the current state of the system and the final state, $J(\mathbf{x}, t)$, then for a safe target set Player 1 will try to minimize the cost while Player 2 will try to maximize the cost and vice versa (Table 3.1).

Thus we modify the system dynamics in (3.1) as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{a}, \mathbf{b}), \tag{3.5}$$

where, $\mathbf{x} \in \mathcal{X}$ is the system state. $\mathbf{a}(\cdot) \in \mathcal{A}$ is the control (action) of Player 1, where $\mathcal{A} \in \mathbb{R}^m$. $\mathbf{b}(\cdot) \in \mathcal{B}$ is the control (action) of Player 2. Since Player 2 is additive noise, the dimension of its control (action) vector is the same as the dimension of the state vector, i.e., $\mathcal{B} \in \mathbb{R}^n$. So the system dynamics described in (3.5) has the form $f : \mathcal{X} \times \mathcal{A} \times \mathcal{B} \to \mathcal{X}$.

Thus the function f has two parts - the system dynamics approximated by a neural network with a single input, which is denoted as $f_{NN}(\mathbf{x}, \mathbf{a})$ in the remaining sections, and the additive bounded noise.

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{a}, \mathbf{b}) = f_{NN}(\mathbf{x} + \mathbf{b}(t), \mathbf{a}). \tag{3.6}$$

Before going into the details of the algorithm and its implementation, we show that all the assumptions in [20] are satisfied for our problem formulation.

Assumption 1:

The control signals $a(\cdot)$ and $b(\cdot)$ for both the players are drawn from compact sets $\mathcal{A} \in \mathbb{R}^m$ and $\mathcal{B} \in \mathbb{R}^n$ respectively, such that:

$$a(\cdot) \in \mathfrak{A}(\mathfrak{t}) \triangleq \{\mu : [0, t] \to \mathcal{A} \mid \mu(\cdot) \text{ is measurable}\},$$

$$b(\cdot) \in \mathfrak{B}(\mathfrak{t}) \triangleq \{\mu : [0, t] \to \mathcal{B} \mid \mu(\cdot) \text{ is measurable}\},$$

where $t \in [0, T]$ for some $T > 0$. Two control signals are considered identical if they agree almost everywhere.

Assumption 2:

The flow field or system dynamics $f_{NN} : \mathcal{X} \times \mathcal{A} \times \mathcal{B} \to \mathcal{X}$ is uniformly continuous, bounded and Lipschitz continuous in $\mathbf{x}$ for fixed values of $\mathbf{a}$ and $\mathbf{b}$.

We need to prove these properties for the neural network that is used to model the system dynamics.

The definition of $tanh()$ is given in Table 3.2. Since its derivative exists over the entire domain, it is continuous and Lipschitz continuous in x when the inputs $\mathbf{a}$ and $\mathbf{b}$ are constant vectors. The output $\hat{\mathbf{o}}$ of the neural network

$$\hat{\mathbf{o}} = c \cdot tanh(W^{(2)} tanh(W^{(1)}\mathbf{x})),$$

is nothing but a nested function of $tanh()$ and is also Lipschitz continuous over the entire domain.

The choice of activation function also determines whether or not the output is bounded. The specific architecture of the neural network used in this work has $tanh()$ activation function for both its hidden layer nodes and output layer nodes. From the properties listed in Table 3.2, it is clear

Table 3.2.
Properties of $tanh()$

Function	Function Definition	Derivative	Domain	Range
tanh	$f(x) = tanh(x) = \frac{(e^x - e^{-x})}{(e^x + e^{-x})}$	$f'(x) = 1 - f(x)^2$	$(-\infty, +\infty)$	$(-1, 1)$

that for an infinite domain $x \in \mathbb{R}$, the range is $tanh(x) \in (-1, 1)$ (Figure 3.3). Thus for a trained neural network, with constant weights $W^{(1)}$ and $W^{(2)}$, the nodes in the hidden layer are computed as

$$\mathbf{h} = tanh(W^{(1)}\mathbf{x})$$

$$\implies |h_i| < 1 \text{ for } i = 1, 2, ...14.$$

Similarly for the output layer

$$\hat{\mathbf{o}} = c \cdot \tanh(W^{(2)}\mathbf{h}).$$

$$\implies |\hat{o}_i| < c \text{ for } i = 1, 2.$$

Hence, the dynamics approximated by the neural network satisfies all the assumptions. Since the noise is also assumed to be bounded, the final function $f(\mathbf{x}, \mathbf{a}, \mathbf{b})$ is bounded. In general, the choice of the activation function determines if the neural network is bounded or not.

Assumption 3:

For the forward reachable set, the *initial set* $\mathcal{X}_0 \subset \mathbb{R}^n$ is closed and can be represented as the zero sublevel set of a bounded and Lipschitz continuous function $g : \mathbb{R}^n \to \mathbb{R}$

$$\mathcal{X}_0 = \{\mathbf{x} \in \mathbb{R}^n | g(\mathbf{x}) \leq 0\} \tag{3.7}$$

For the test problem, the initial set is a circular region and is modelled using the following function definition

$$g(\mathbf{x}) = x^2 + y^2 - r_0^2. \tag{3.8}$$

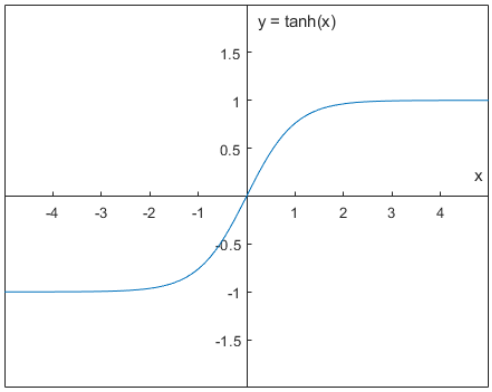

Fig. 3.3. Evolution of $tanh(x)$ where $x \in \mathbb{R}$.

The above assumptions are necessary to prove the properties of the reachable set.

Trajectory

We now formally define a trajectory. If we fix the values for both the control inputs $a(\cdot) \in \mathfrak{A}(t)$, $b(\cdot) \in \mathfrak{B}(t)$, then for a given initial point $\mathbf{x_0}$, there exists a unique trajectory solving (3.6).

$$\xi_f(s; \mathbf{x_0}, 0, \mathbf{a}(\cdot), \mathbf{b}(\cdot)) : [0, t] \to \mathbb{R}^n, \tag{3.9}$$

where $s \in [0,\ t]$.

The definition in (3.9) must satisfy the initial condition $\xi_f(0; \mathbf{x_0}, 0, \mathbf{a}(\cdot), (\mathbf{b}(\cdot))) = \mathbf{x_0}$ and the following differential equation almost everywhere

$$\frac{d}{ds}\xi_f(s; \mathbf{x}, 0, \mathbf{a}(\cdot), \mathbf{b}(\cdot)) = f(\xi_f(s; \mathbf{x}, 0, \mathbf{a}(\cdot), \mathbf{b}(\cdot)), \mathbf{a}(\mathbf{s}), \mathbf{b}(\mathbf{s})).$$

Information strategies for Player 1 and Player 2

For a differential game, it is necessary to define the information strategies for both players. The information strategy states what information is available to each player about the other player's strategy before taking an action at any time instant. Reachable sets aim at giving a conservative solution, so it is natural that the strategies for both the players should be designed to over-approximate the forward reachable set for a safe target set in the presence of noise.

Thus, the information strategy is defined in such a way that the adversary, Player 2, gets an advantage by following a *non-anticipative strategy* which is defined as a map $\gamma : \mathfrak{A}(t) \to \mathfrak{B}(t)$. The control taken by Player 2 is a function of that taken by Player 1 and the mapping is defined as

$$\begin{aligned} \gamma \in \Gamma(t) \triangleq \{\rho : \mathfrak{A}(t) \to \mathfrak{B}(t) | a(r) = \hat{a}(r) \\ \text{for almost every} \quad r \in [t, s] \\ \implies \rho[a](r) = \rho[\hat{a}](r) \\ \text{for almost every } r \in [t, s]\} \end{aligned} \tag{3.10}$$

The above equation means that if Player 2 cannot distinguish between control signals $a(\cdot)$ and $\hat{a}(\cdot)$ of Player 1 until after time s, then Player 2 will use the same control signal until it can distinguish between the two.

Essentially, this implies that Player 2 is given the advantage to select a control input after considering the best case scenario (maximize the cost function $J(\mathbf{x}, t)$), for all possible values of Player 1's input. After Player 1 has taken an action, Player 2 observes his action and selects a control to maximize the cost function. Mathematically, this gives a *minmax* optimization problem.

$$\min_{a \in A} \max_{b \in B} J(\mathbf{x}, t). \tag{3.11}$$

In the following sections, we will see how the above formulation is adapted into an HJI equation.

From the above information, we can now define the reachable set mathematically. For completion, we provide the definition of both forward and backward reachable sets

Forward Reachable Set: The forward reachable set for a finite time $t = T$ and initial set $\mathcal{X}_0$ is defined as

$$\begin{aligned} \mathcal{W}(\tau) \triangleq \{w \in \mathbb{R}^n | \exists \gamma \in \Gamma(t), \forall a(\cdot) \in \mathcal{A}(t), \exists s \in [0, t], \\ \xi_f(s; x, 0, a(\cdot), \gamma[a](\cdot)) = w, x \in \mathcal{X}_0\}, \end{aligned} \tag{3.12}$$

where, $\tau \in [0, t]$.

Backward Reachable Set: The backward reachable set for a finite time trajectory $t = -T$ and target set $\mathcal{X}_0$ is defined as

$$\begin{aligned} \mathcal{V}(\tau) \triangleq \{v \in \mathbb{R}^n | \exists \gamma \in \Gamma(t), \forall a(\cdot) \in \mathcal{A}(t), \exists s \in [t, 0], \\ \xi_f(s; x, t, a(\cdot), \gamma[a](\cdot)) \in \mathcal{X}_0\}, \end{aligned} \tag{3.13}$$

where, $\tau \in [t, 0]$.

The above definitions must satisfy the closed set property of reachable set. Indeed, in [20], Dr. Ian Mitchell stated and proved the following

Theorem: If $\mathcal{X}_0$ is closed, then $\mathcal{V}(\tau)$ is closed. Conversely, if $\mathcal{X}_0$ is open, then $\mathcal{V}(\tau)$ is open.

The above theorem was proved by using the fact that if $\mathcal{X}_0$ is closed, then $\mathcal{X}_0^C = \mathbb{R}^n \backslash \mathcal{X}_0$ is open. Thus, it suffices to show that $\mathcal{V}(\tau)^C$ is open for $\tau \in [-T, 0]$, and consequently, $\mathcal{V}(\tau)$ is closed. The fact the learned system dynamics is lipschitz continuous is used to bound the growth rate of trajectories in the open space $\mathcal{X}_0^C$. Interested readers can refer to the detailed proof in [20].

3.2.2 Level Set Method

In the previous section, we formulated the control strategy for Player 1 and Player 2 to compute the reachable set in (3.11) and showed that this reachable set is always closed as it grows with time. Now, we need an efficient way to represent this reachable set such that we can keep track of its continuous boundary and perform set operations (like union and intersection) easily.

This section aims at describing the basic idea behind the level set method for surface propagation [52]. We will not be delving into the details of level set method but give enough context to understand how it is being utilized to compute reachable sets.

The level set method is a numerical technique which is used to track the propagation of surfaces. Usually the starting surface has a well defined geometric shape but the level set method is capable of tracking any irregularly shaped surfaces over time. An illustrative example of a circular surface propagating with constant speed is given in Figures 3.4 and 3.5. This is achieved by modelling the evolution of the boundary/interface of the surface when it is moving with a speed F in the normal direction. This is a very useful property which can be leveraged while computing reachable sets.

Let the initial position of the interface/boundary of the surface be denoted as Υ, where Υ is a closed curve in $\mathbb{R}^n$. If this interface is moving with a speed F normal to it, then we define a function $\phi(\mathbf{x}, t = 0)$ from $\mathbb{R}^n \rightarrow \mathbb{R}$ such that Υ is its zero level set and the distance of any point $\mathbf{x}$ from the interface is $\phi(\mathbf{x}, t = 0) = +d$ if it lies outside the interface and $\phi(\mathbf{x}, t = 0) = -d$ if it lies within the interface.

Now, an evolution equation can be derived for the above interface by differentiating it with respect to time t. Using the chain rule, this equation comes out as

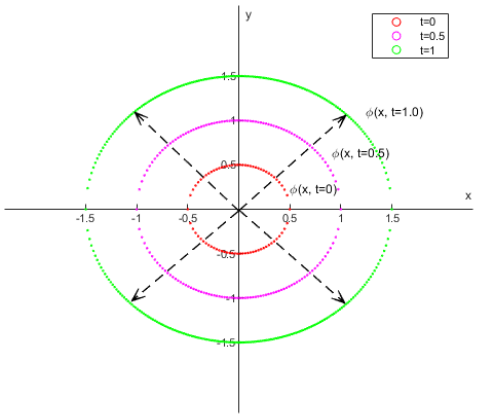

Fig. 3.4. Cross-sectional view of a circular surface $\phi(\mathbf{x}, t = 0)$ propagating with constant speed.

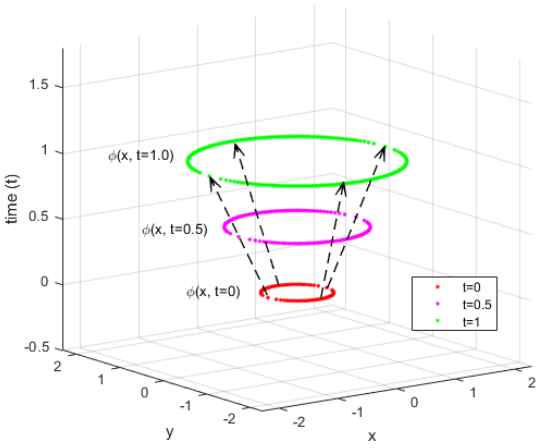

Fig. 3.5. 3D view of the initial propagating surface $\phi(\mathbf{x}, t = 0)$ at different time instances.

$$\frac{\partial \phi}{\partial t} + \frac{\partial \phi}{\partial \mathbf{x}} \cdot \frac{\partial \mathbf{x}}{\partial t} = 0, \tag{3.14}$$

or for the ease of notation

$$\begin{aligned} D_t \phi(\mathbf{x}, t) + F|D_{\mathbf{x}} \phi(\mathbf{x}, t)| &= 0, \\ \phi(\mathbf{x}, t = 0) &= given \end{aligned} \tag{3.15}$$

Equation (3.15) is an initial valued partial differential equation which should be solved for ϕ at any time t to get the position of the interface at that instant. Numerical schemes approximating the gradient of function ϕ are used to compute the solution at any given time.

3.2.3 Computation of Reachable Set using Level Set Method

In (3.15), if the speed F is replaced by the system dynamics and the gradient of function ϕ is written as a vector $\mathbf{p} \in \mathbb{R}^n$, then the equation can be rewritten as

$$D_t\phi(\mathbf{x}, t) + \mathbf{p}^T f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b}) = 0,$$

$$\phi(\mathbf{x}, t = 0) = g(\mathbf{x}).$$

The term $\mathbf{p}^T f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b})$ governs the speed of propagation of the reachable set, i.e., by what amount the boundary of the set moves forward from time t to $t + \Delta t$. To compute the forward reachable set, the speed should not be allowed to be negative.

$$D_t\phi(\mathbf{x}, t) + max[0, \mathbf{p}^T f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b})] = 0, \tag{3.16}$$

The strategy to determine the values of control inputs a and b have already been define in (3.11). Now, the cost function is the term $\mathbf{p}^T f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b})$ and we define the Hamiltonian as

$$H(\mathbf{x}, \mathbf{p}) = \min_{a \in A} \max_{b \in B} \mathbf{p}^T f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b}). \tag{3.17}$$

The above equation represents a general form of the Hamiltonian function. However, to analyze the behavior of a particular controller, the control input for Player 1 should be defined by the control law of the controller.

General Problem Formulation

The reachable set of a Model-Based RL controller with deterministic dynamics $f_{NN} : \mathcal{X} \times \mathcal{A} \times \mathcal{B} \to \mathcal{X}$ and following the control law $\pi : \mathcal{X} \to \mathcal{A}$, is determined by computing the *viscosity solution* ϕ of the following time-dependent HJI equation

$$\begin{aligned} D_t\phi(\mathbf{x}, t) + max[0, H_{const}(\mathbf{x}, D_{\mathbf{x}}\phi(\mathbf{x}, t))] = 0, \\ \phi(\mathbf{x}, t = 0) = g(\mathbf{x}) \end{aligned} \tag{3.18}$$

where,

$$H_{const}(\mathbf{x}, \mathbf{p}) = \min_{a \in A} \max_{b \in B} \mathbf{p}^T f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b}) \tag{3.19}$$

subject to

$$\mathbf{a} = \pi(\mathbf{a}|\mathbf{x}).$$

For the neural network approximated system dynamics with weights $W_f^{(1)}$ and $W_f^{(2)}$ used in the test problem, the function $f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b})$ is computed as

$$f_{NN}(\mathbf{x}, \mathbf{a}, \mathbf{b}) = c \cdot tanh(W_f^{(2)} tanh(W_f^{(1)} \cdot (\mathbf{x} + \mathbf{b}))), \tag{3.20}$$

and the controller parameterized by weights $W_c^{(1)}$ and $W_c^{(2)}$ can be substituted as

$$\mathbf{a} = tanh(W_c^{(2)} \cdot tanh(W_c^{(1)} \cdot \mathbf{x})) \tag{3.21}$$

Viscosity Solution Viscosity solution is a unique, weak solution to a PDE when its classical solution may not exist [51, 53, 54]. This viscosity solution also represents the value of the differential game in (3.19). This statement is proved in [20] for the backward reachable set from an unsafe target set. It is shown that for a game with terminal cost

$$C(x, t, a(\cdot), b(\cdot)) = g(\xi_f(0; x, t, a(\cdot), b(\cdot))),$$

and no running cost, Player 1 will be trying to maximize the terminal cost, while Player 2 will try to minimize this cost. Thus, the value function of the resulting differential game is given as

$$v(x, t) = \inf_{\gamma \in \Gamma(t)} \sup_{a(\cdot) \in \mathcal{A}(t)} C(x, t, a(\cdot), b(\cdot))$$

$$= \inf_{\gamma \in \Gamma(t)} \sup_{a(\cdot) \in \mathcal{A}(t)} g(\xi_f(0; x, t, a(\cdot), \gamma[a](\cdot))).$$

Theorem 4.1 in [51] shows that this value function is the viscosity solution to the HJI PDE above. In addition, [20] details the proof to show that this value function is same as the reachable set. The viscosity solution is computed using the well-developed numerical tools in the level set method literature.

Thus, the following conclusion can be made.

Corollary : If for a given Model-Based RL controller, with deterministic system dynamics f_{NN} and control law π, the forward reachable set $\mathcal{W}(T)$, starting from an initial set $\mathcal{X}_0$, is computed by finding the viscosity solution for (3.18), then for any state $\mathbf{x} \in \mathcal{W}(T)$ and unsafe region $\psi \in \mathbb{R}^n$:

$$\Big|\{\mathbf{x} : \mathbf{x} \in \mathcal{W}(T) \;\; and \;\; \mathbf{x} \in \psi\}\Big| > 0,$$

then the controller fails to meet the safety constraints.

3.2.4 Implementation

The reachable set at time $t + \Delta t$, ϕ can be computed as

$$\phi^{t+\Delta t} = \phi^t - \Delta t \cdot max[0, H_{const}(\mathbf{x}, D_{\mathbf{x}}\phi(\mathbf{x}, t))]. \tag{3.22}$$

The above algorithm was implemented to compute the reachable set for the test problem. The toolbox developed by Dr. Ian Mitchell [21] is used to obtain the solution of the reachable set. This toolbox comes with inbuilt numerical schemes to compute the approximation for the Hamiltonian H_{const} and spacial derivative.

In the current implementation, a Lax-Friedrichs approximation to compute the Hamiltonian is used [55]. Numerical schemes such as fifth order accurate weighted essentially nonoscillatory approximation [56] and the Runge-Kutta scheme [57] is used to compute the spatial derivative and time integration.

3.3 Results

The above formulation of the reachability problem was solved for the test problem. Unfortunately, the implementation of the forward reachable set is not as fine which is why the algorithm is over-approximating the reachable set by a significant margin. It should be noted that this over-

approximation is not due to theoretical results but a draw back of the current implementation. Results are presented for cases with and without the addition of noise.

Case 1: Safety verification in the absence of noise

We first present the reachable set in the absence of noise. The results from the reachabality analysis show that even in the absence of noise, the controller fails to satisfy the constraints throughout a finite time trajectory as seen in Figure 3.3. This observation is further verified by sampling trajectories from the boundary of the initial set and is displayed in Figure 3.7. This highlights a major drawback of reinforcement learning controllers. If there is no mechanism to handle hard constraints in the planning algorithm then it is very difficult to comment on the safety guarantees of a controller without a verification tool.

Thus, the proposed framework was successfully able to identify if the controller is safe under the given initial conditions or not.

Case 2: Safety verification in the presence of bounded noise

The reachable set was also computed in the presence of bounded noise. Figure 3.3 shows the plots for the forward reachable set computed in the presence of noise $\mathcal{N}_t \in [-0.05,\ +0.05]$. Figure 3.3 shows the same plots for noise $\mathcal{N}_t \in [-0.1,\ +0.1]$. Since the controller failed to perform safely in the absence of noise, it is clear that it will also violate the constraints in the presence of noise. This is evident from the growth of the reachable set seen in figures below. From all the set of results, it is clear that the current numerical implementation is over-approximating the reachable set by a significant margin. The fact that the area covered by the reachable set in presence of noise is more than that in the absence of noise implies that the over-approximation error is consistent and relatively, the performance of the controller degrades further in the presence of noise.

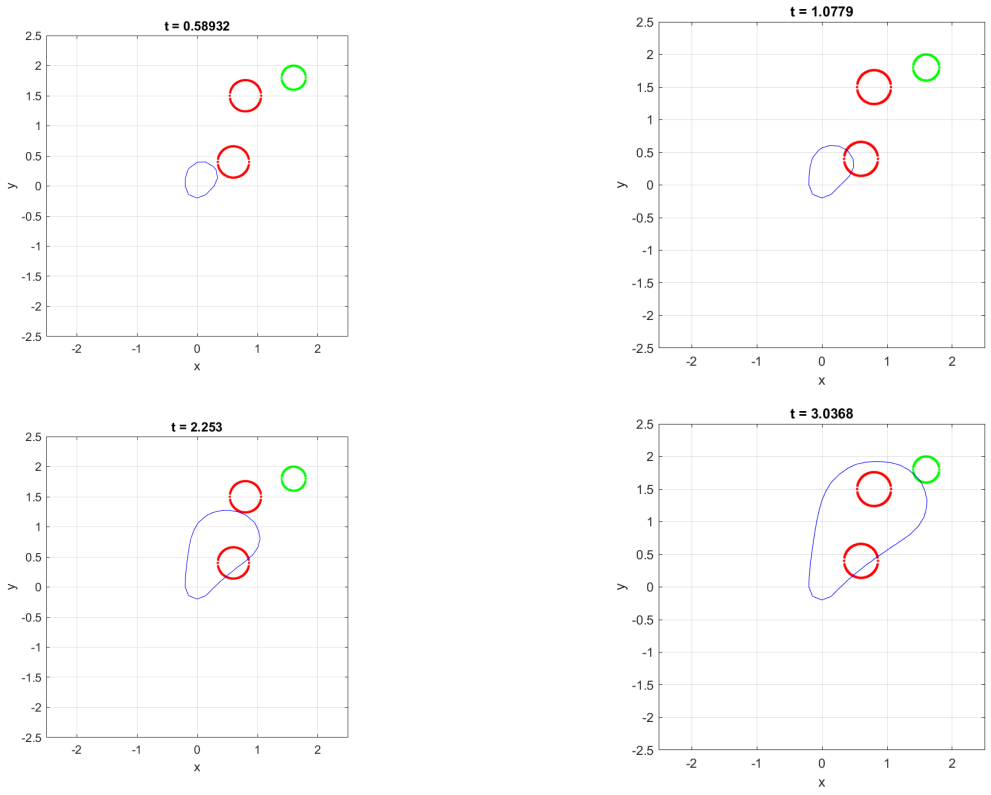

Fig. 3.6. Forward reachable set computed in the absence of noise for the test problem. The results imply that even in the absence of noise the Model-Based RL controller doesn't always satisfy the constraints in the state space.

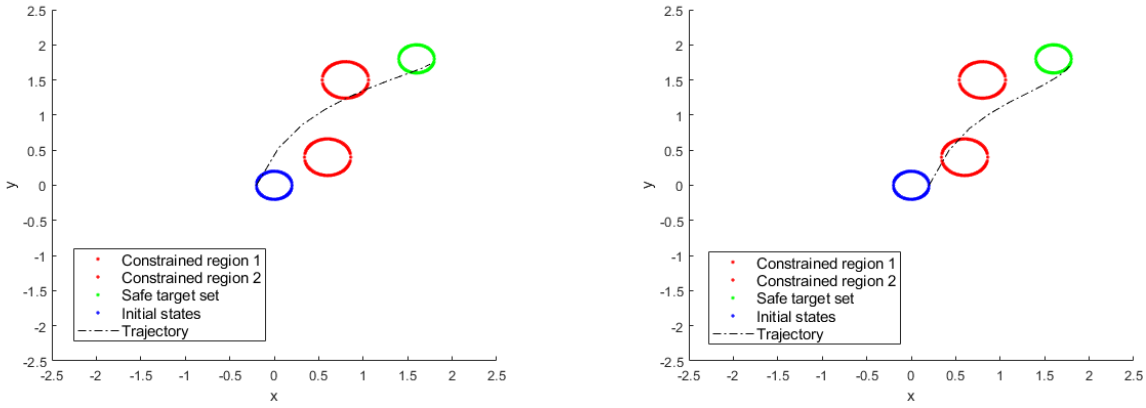

Fig. 3.7. Trajectories sampled from initial points $(-0.2, 0.0)$ and $(0.2, 0.0)$ violate the state constraints in the absence of noise.

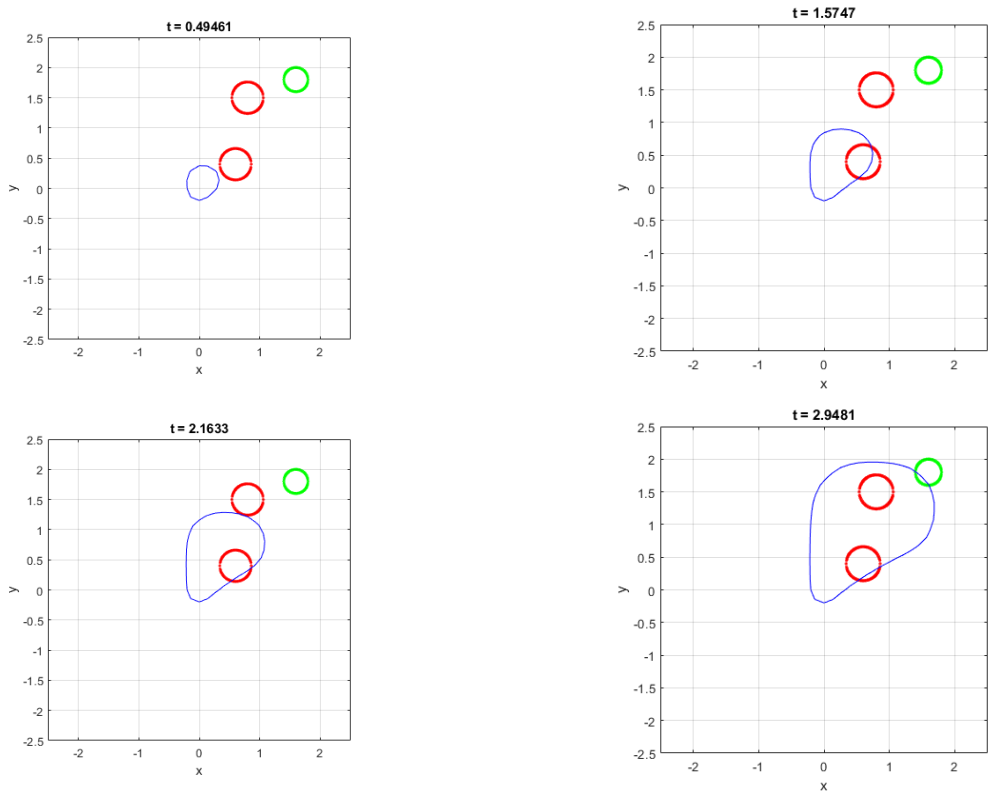

Fig. 3.8. Forward reachable set computed in the presence of bounded noise $\mathcal{N}_t \in [-0.05,\ +0.05]$ for the test problem.

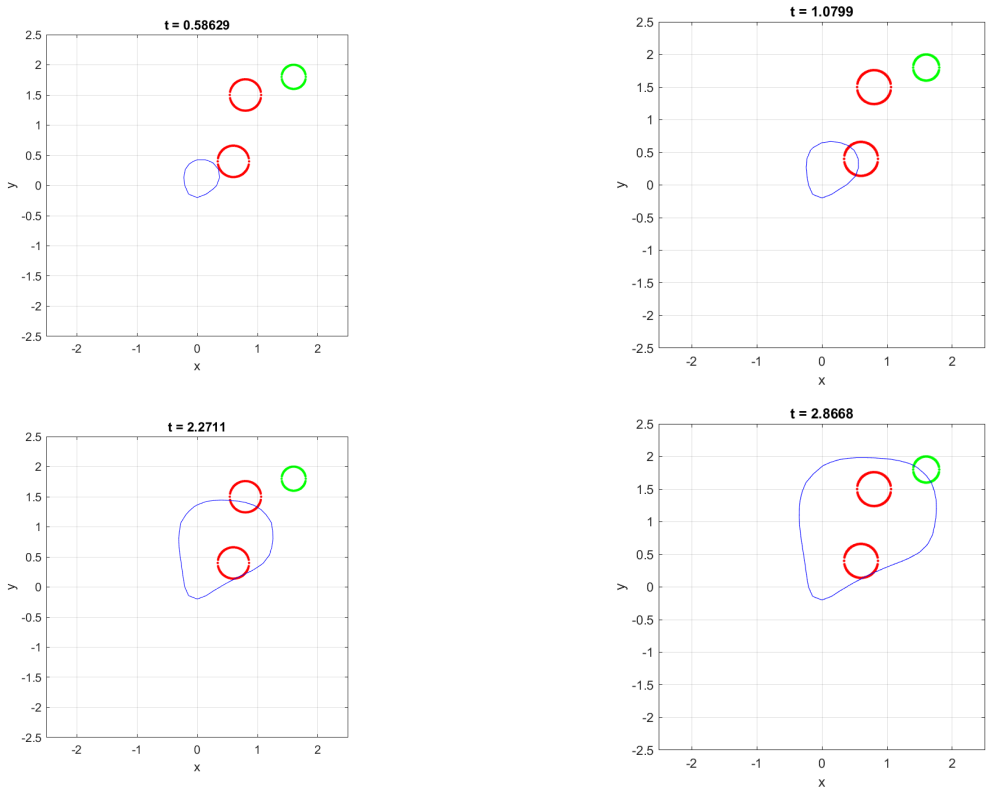

Fig. 3.9. Forward reachable set computed in the presence of bounded noise $\mathcal{N}_t \in [-0.1,\ +0.1]$ for the test problem.

4. SUMMARY

The work presented in this thesis covers the theoretical formulation of the problem of reachable set computation for Model-Based RL controllers in the presence of bounded noise. Specifically, we have presented the mathematical formulation for computing the reachable set for a system whose dynamics is learned by a neural network. By computing the forward reachable set for such a controller and analyzing its interaction with the system constraints a binary yes or no answer can be given to state whether the trained controller will continue to satisfy the constraints or not in the presence of noise for the given set of initial conditions.

The HJI-formulation for computing the reachable set was suitable for the problem statement in this thesis. However, this method has been known to have issues with scaling dimension of the state vector. Also, the accuracy of the final reachable set depends on the implementation of the algorithm. The discretization error compounds over time giving a significantly over-approximated reachable set in implementation.

The current work successfully provides a general framework for the development of a verification tool to evaluate the safety of a Model-Based RL controller with deterministic system dynamics.

5. FUTURE WORK

The method proposed in this work is proved to be theoretically sound for the problem at hand. However, there is still room for improvement.

1. *Extending for stochastic system dynamics:* One scope for improvement is to extend this algorithm for stochastic system dynamics. The same argument can be made for a stochastic controller which returns a possible set of control inputs instead of a single deterministic control input.

2. *Giving expected performance in terms of the reward function:* The current work gives a yes or no answer to whether the given initial states are safe to start propagating trajectories in the presence of bounded noise. While this is an important guarantee for safety critical systems, a more general statement will be to provide some kind of performance metric in terms of the expected reward computed over the entire space of the reachable set (Figure 5.1).

3. *Computing backward reachable set:* To compute the backward reachable set, the system dynamics must be reversible. However, this property has not been proven to hold for neural

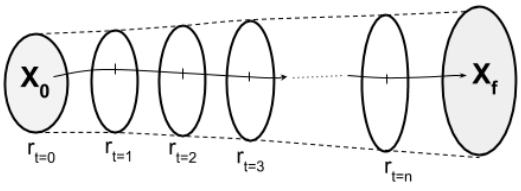

Fig. 5.1. Compute the expected reward over a surface at every time step to give a performance metric for the controller in the presence of noise.

networks. Computing the backward reachable set from an unsafe target set can give the set of initial states to avoid while propagating trajectories. This is practically a more useful result.

Bibliography

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[2] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[3] N. Golovin and E. Rahm, "Reinforcement learning architecture for web recommendations," in *International Conference on Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004.*, vol. 1. IEEE, 2004, pp. 398–402.

[4] A. Karatzoglou, L. Baltrunas, and Y. Shi, "Learning to rank for recommender systems," in *Proceedings of the 7th ACM conference on Recommender systems*. ACM, 2013, pp. 493–494.

[5] L. Wang, W. Zhang, X. He, and H. Zha, "Supervised reinforcement learning with recurrent neural network for dynamic treatment recommendation," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 2018, pp. 2447–2456.

[6] I. Szita, "Reinforcement learning in games," in *Reinforcement Learning*. Springer, 2012, pp. 539–577.

[7] L. Bom, R. Henken, and M. Wiering, "Reinforcement learning to train ms. pac-man using higher-order action-relative inputs," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*. IEEE, 2013, pp. 156–163.

[8] G. Boone, "Efficient reinforcement learning: Model-based acrobot control," in *Proceedings of International Conference on Robotics and Automation*, vol. 1. IEEE, 1997, pp. 229–234.

[9] H. Benbrahim and J. A. Franklin, "Biped dynamic walking using reinforcement learning," *Robotics and Autonomous Systems*, vol. 22, no. 3-4, pp. 283–302, 1997.

[10] G. Endo, J. Morimoto, T. Matsubara, J. Nakanishi, and G. Cheng, "Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 213–228, 2008.

[11] J. Morimoto and K. Doya, "Acquisition of stand-up behavior by a real robot using hierarchical reinforcement learning," *Robotics and Autonomous Systems*, vol. 36, no. 1, pp. 37–51, 2001.

[12] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, 2018.

[13] J. H. Gillula and C. J. Tomlin, "Guaranteed safe online learning via reachability: tracking a ground target using a quadrotor," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 2723–2730.

[14] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[15] T. J. Perkins and A. G. Barto, "Lyapunov design for safe reinforcement learning," *Journal of Machine Learning Research*, vol. 3, no. Dec, pp. 803–832, 2002.

[16] L. Wang, E. A. Theodorou, and M. Egerstedt, "Safe learning of quadrotor dynamics using barrier certificates," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 2460–2465.

[17] T. J. Walsh, S. Goschin, and M. L. Littman, "Integrating sample-based planning and model-based reinforcement learning," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[18] H. Xu, Y. Li, Y. Tian, T. Darrell, and T. Ma, "Algorithmic framework for model-based reinforcement learning with theoretical guarantees," *arXiv preprint arXiv:1807.03858*, 2018.

[19] Z. Zuo, Z. Wang, Y. Chen, and Y. Wang, "A non-ellipsoidal reachable set estimation for uncertain neural networks with time-varying delay," *Communications in Nonlinear Science and Numerical Simulation*, vol. 19, no. 4, pp. 1097–1106, 2014.

[20] I. M. Mitchell, "Application of level set methods to control and reachability problems in continuous and hybrid systems." 2003.

[21] ——, "A toolbox of level set methods," *UBC Department of Computer Science Technical Report TR-2007-11*, 2007.

[22] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[23] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[24] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[25] L. C. Baird III and A. W. Moore, "Gradient descent for general reinforcement learning," in *Advances in neural information processing systems*, 1999, pp. 968–974.

[26] O. Maler, "Computing reachable sets: An introduction," *Tech. rep. French National Center of Scientific Research*, 2008.

[27] O. Stursberg and B. H. Krogh, "Efficient representation and computation of reachable sets for hybrid systems," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2003, pp. 482–497.

[28] A. Girard, C. Le Guernic, and O. Maler, "Efficient computation of reachable sets of linear time-invariant systems with inputs," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2006, pp. 257–271.

[29] T. Dang and O. Maler, "Reachability analysis via face lifting," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 1998, pp. 96–109.

[30] E. K. Kostousova, "External polyhedral estimates of reachable sets of linear and bilinear discrete-time systems with integral bounds on additive terms," in *2018 14th International Conference" Stability and Oscillations of Nonlinear Control Systems"(Pyatnitskiy's Conference)(STAB)*. IEEE, 2018, pp. 1–4.

[31] ——, "On polyhedral estimates for reachable sets of discrete-time systems with bilinear uncertainty," *Automation and Remote Control*, vol. 72, no. 9, p. 1841, 2011.

[32] A. B. Kurzhanski and P. Varaiya, "Ellipsoidal techniques for reachability analysis," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2000, pp. 202–214.

[33] T. F. Filippova, "Description of dynamics of ellipsoidal estimates of reachable sets of nonlinear control systems with bilinear uncertainty," in *International Conference on Numerical Methods and Applications*. Springer, 2018, pp. 97–105.

[34] Y. Chen and J. Lam, "Estimation and synthesis of reachable set for discrete-time periodic systems," *Optimal Control Applications and Methods*, vol. 37, no. 5, pp. 885–901, 2016.

[35] O. Stursberg and B. H. Krogh, "Efficient representation and computation of reachable sets for hybrid systems," in *International Workshop on Hybrid Systems: Computation and Control*. Springer, 2003, pp. 482–497.

[36] J. Lin, "Determination of reachable set for a linear discrete system," *IEEE Transactions on Automatic Control*, vol. 15, no. 3, pp. 339–342, 1970.

[37] S. Bogomolov, M. Forets, G. Frehse, F. Viry, A. Podelski, and C. Schilling, "Reach set approximation through decomposition with low-dimensional sets and high-dimensional matrices," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week)*. ACM, 2018, pp. 41–50.

[38] S. Kaynama and M. M. K. Oishi, "Schur-based decomposition for reachability analysis of linear time-invariant systems," *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pp. 69–74, 2009.

[39] M. Chen, S. L. Herbert, M. S. Vashishtha, S. Bansal, and C. J. Tomlin, "Decomposition of reachable sets and tubes for a class of nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 63, no. 11, pp. 3675–3688, 2018.

[40] Y. Lin and S. Saripalli, "Sampling-based path planning for uav collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, vol. 18, no. 11, pp. 3179–3192, 2017.

[41] H.-T. Chiang, N. Malone, K. Lesser, M. Oishi, and L. Tapia, "Path-guided artificial potential fields with stochastic reachable sets for motion planning in highly dynamic environments," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015, pp. 2347–2354.

[42] Y. Lin and S. Saripalli, "Collision avoidance for uavs using reachable sets," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE, 2015, pp. 226–235.

[43] Y. Zhou and J. S. Baras, "Reachable set approach to collision avoidance for uavs," in *2015 54th IEEE Conference on Decision and Control (CDC)*. IEEE, 2015, pp. 5947–5952.

[44] N. Malone, H.-T. Chiang, K. Lesser, M. Oishi, and L. Tapia, "Hybrid dynamic moving obstacle avoidance using a stochastic reachable set-based potential field," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1124–1138, 2017.

[45] J. H. Gillula, G. M. Hoffmann, H. Huang, M. P. Vitus, and C. J. Tomlin, "Applications of hybrid reachability analysis to robotic aerial vehicles," *The International Journal of Robotics Research*, vol. 30, no. 3, pp. 335–354, 2011.

[46] M. Chen, J. C. Shih, and C. J. Tomlin, "Multi-vehicle collision avoidance via hamilton-jacobi reachability and mixed integer programming," in *2016 IEEE 55th Conference on Decision and Control (CDC)*. IEEE, 2016, pp. 1695–1700.

[47] M. Chen, J. F. Fisac, S. Sastry, and C. J. Tomlin, "Safe sequential path planning of multi-vehicle systems via double-obstacle hamilton-jacobi-isaacs variational inequality," in *2015 European Control Conference (ECC)*. IEEE, 2015, pp. 3304–3309.

[48] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 1424–1431.

[49] M. Chen, S. Bansal, J. F. Fisac, and C. J. Tomlin, "Robust sequential path planning under disturbances and adversarial intruder," *arXiv preprint arXiv:1611.08364*, 2016.

[50] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-jacobi reachability: A brief overview and recent advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2242–2253.

[51] L. C. Evans and P. E. Souganidis, "Differential games and representation formulas for solutions of hamilton-jacobi-isaacs equations," *Indiana University mathematics journal*, vol. 33, no. 5, pp. 773–797, 1984.

[52] J. A. Sethian, "A fast marching level set method for monotonically advancing fronts," *Proceedings of the National Academy of Sciences*, vol. 93, no. 4, pp. 1591–1595, 1996.

[53] M. G. Crandall and P.-L. Lions, "Viscosity solutions of hamilton-jacobi equations," *Transactions of the American mathematical society*, vol. 277, no. 1, pp. 1–42, 1983.

[54] M. G. Crandall, L. C. Evans, and P.-L. Lions, "Some properties of viscosity solutions of hamilton-jacobi equations," *Transactions of the American Mathematical Society*, vol. 282, no. 2, pp. 487–502, 1984.

[55] M. G. Crandall and P.-L. Lions, "Two approximations of solutions of hamilton-jacobi equations," *Mathematics of computation*, vol. 43, no. 167, pp. 1–19, 1984.

[56] S. Osher and C.-W. Shu, "High-order essentially nonoscillatory schemes for hamilton–jacobi equations," *SIAM Journal on numerical analysis*, vol. 28, no. 4, pp. 907–922, 1991.

[57] C.-W. Shu and S. Osher, "Efficient implementation of essentially non-oscillatory shock-capturing schemes," *Journal of computational physics*, vol. 77, no. 2, pp. 439–471, 1988.